

From Bit to Bitcoin



Understanding Blockchain
from First Principles

01010.01100

1

Charles D. Cheerful

Contents

0 Chapter 0: Welcome to the Journey	11
0.1 What This Book Is	11
0.2 What You'll Learn	11
0.3 A Warning About Propaganda	11
0.4 Why Learn This?	13
0.5 A Note on Tone	13
0.6 A Note on Data Accuracy and AI Usage	13
0.7 The Promise	13
0.8 Let's Begin	14
1 Chapter 1: The Bit	15
1.1 The Simplest Possible Information	15
1.2 From Light Switches to Computers	15
1.3 Binary: The Language of Two	16
1.4 Eight Switches = One Byte	16
1.5 All Complexity Builds From This	17
1.6 Under the Hood: How Computers Store a Bit, Like Physically	18
2 Chapter 2: Logos - Mapping Meaning to Numbers	19
2.1 The Meaning Problem	19
2.2 The ASCII Table: A Social Agreement	19
2.3 My Binary BRO	20
2.4 Beyond Letters: Everything is Numbers	21
2.5 The Deep Insight: Information is Agreement	21
2.6 Languages Work the Same Way	22
2.7 Why This Matters for Bitcoin	22
3 Chapter 3: Algorithms - Following Rules	25
3.1 The Recipe Analogy	25
3.2 A Simple Algorithm: Binary to Letters	26
3.3 Computers Don't "Think"	26
3.4 The Power of Determinism	27
3.5 Algorithms Can Be Simple or Complex	27
3.6 Computers Follow Recipes Perfectly (and Stupidly)	28
3.7 Why This Matters for Bitcoin	28
3.8 The Transition to Protocols	29
4 Chapter 4: Protocols - Computers Talking	31
4.1 The Allergy Protocol	31
4.2 The Internet is Just Protocols	32
4.3 Protocols are Everywhere	33

4.4	The Internet Protocol (IP)	33
4.5	Bitcoin is a Protocol Too	34
4.6	Protocols Enable Trust Without Authority	34
4.7	Why Protocols Matter for Bitcoin	35
4.8	The Network Effect	35
4.9	Protocols are Living Standards	36
5	Chapter 5: The Trust Problem	37
5.1	Alice, Bob, and Carol	38
5.2	Historical Solutions	38
5.3	The Fundamental Question	39
5.4	Why This Matters Today	39
5.5	The Solution: Encryption	39
5.6	The Setup for What's Next	40
6	Chapter 6: Symmetric Encryption - Shared Secrets	43
6.1	The Caesar Cipher	43
6.2	The Password Principle	44
6.3	Why It's Called "Symmetric"	45
6.4	Modern Symmetric Encryption	46
6.5	The Fatal Flaw	46
6.6	Long-Distance Relationships Did Not Work	47
6.7	Why This Matters for Bitcoin	47
7	Chapter 7: Asymmetric Encryption - The Magic Trick and How They Create Crypto-Wallets	51
7.1	The One-Way Function	51
7.2	An Example For The Curious: Multiplication of Prime Numbers	52
7.3	The Two Keys: Public and Private	53
7.4	How It Works	54
7.5	The Real Magic: RSA Encryption	55
7.6	Why This Matters for Bitcoin	55
7.7	Cool Properties	57
7.8	What We've Solved	58
7.9	Summary	58
8	Chapter 8: Quantum Computing - The Future Threat?	59
8.1	The Concern	59
8.2	What is Quantum Computing?	59
8.3	What Breaks and What Doesn't	60
8.4	Why Cryptocurrencies Can Adapt	60
8.5	Everyone Has This Problem	60
8.6	The Treasure Hunt: Satoshi's Bitcoin	60
9	Chapter 9: Money as Synchronized Bits	63
9.1	What Makes Something Money?	63
9.1.1	Property 1: Scarcity (You Can't Create It Easily)	63
9.1.2	Property 2: Verifiability (You Can Prove What You Have)	64

9.1.3	Property 3: No Double-Spending (Can't Spend the Same Money Twice)	64
9.1.4	Property 4: Transferability (You Can Send It)	64
9.1.5	Property 5: Ownership (You Actually Control It)	64
9.1.6	Property 6: Fungibility (Each Unit is Equivalent)	65
9.1.7	Property 7: Divisibility (Can Break Into Smaller Parts)	65
9.1.8	Property 8: Durability (Doesn't Disappear or Decay)	65
9.2	The Realization: Bits Can Have These Properties	65
9.3	The Database Problem	66
9.4	The Traditional Solution: One Database, One Controller	66
9.5	The Insight: Distribute the Database	66
9.6	The Consensus Problem	67
9.7	The Questions We Need to Answer	68
9.8	Historically Speaking	69
10	Chapter 10: Proof-of-Work - Earning the Right to Write	71
10.1	Borrowing Hierarchy (Just for a Moment)	71
10.2	Who Gets the Right to Write?	72
10.3	How Humans Choose Leaders	72
10.4	The First Digital Feat: Proof-of-Work	72
10.5	The Puzzle	73
10.6	Why This Works	74
10.7	The Incentive: Mining Rewards	75
10.8	The Miracle of Bitcoin, the Wet Dreams of Nerds, Maybe	75
10.9	Temporary Leadership	76
10.10	What We've Solved (Have We?) and What We Haven't	76
10.11	The Bigger Picture	76
10.12	What is a Blockchain, Really?	77
10.13	A Note on Satoshi's Thinking	78
11	Chapter 11: The Blockchain - Chaining History Together	79
11.1	The Simultaneous Solution Problem	79
11.2	The Impossible Choice	80
11.3	The Elegant Solution: Let Them Compete	80
11.4	For How Long Do We Keep Doing This?	81
11.5	Understanding Finality	82
11.6	Now, What Is a Blockchain, Actually?	83
11.7	Optional: For the Curious (You Can Skip This)	83
12	Chapter 12: The Blockchain Data Structure (Technical Deep-Dive)	85
12.1	How Does the Blockchain Data Structure Actually Work?	85
12.1.1	What Is a Data Structure? And What Is a Computer's Memory?	85
12.1.2	Pointers: References to Other Locations	86
12.1.3	Linked Lists: Chaining Data Together	87
12.1.4	Hash Functions (Review)	87
12.2	Putting It All Together: The Blockchain Data Structure	88
12.3	Why This Structure Matters: Tamper-Evident History	88
12.4	But Can't You Just Recompute All the Hashes?	89
12.5	The Longest Chain Rule (Refreshser)	89

12.6 The Anti-Gaslight Machine	90
12.7 The 51% Attack	91
12.8 The Blockchain Structure Summarized	91
12.9 Why This Matters	92
13 Chapter 13: Ethereum - The Consensual Computing Machine	93
13.1 Bitcoin Recap: Simple Transactions	93
13.2 The Insight: Programmable Transactions	94
13.3 Smart Contracts: Consensual Logic On Code	94
13.3.1 Example 1: Loans with Collateral	94
13.3.2 Example 2: Will (Inheriting Bits)	95
13.3.3 Example 3: Subscription (Automatic Recurring Payments)	95
13.4 The Ethereum Virtual Machine (EVM): Everyone Runs the Same Programs	96
13.4.1 Addresses: Still Based on Asymmetric Keys	96
13.4.2 How It Works:	96
13.5 Why This Is Revolutionary Innovation: Unstoppable Programs	97
13.6 What Could This Enable? Let Your Imagination Run	97
13.7 The Downsides:	98
13.7.1 1. You Have to Pay for It	98
13.7.2 2. You can build anything! Wait... anything?	98
13.7.3 3. Code Can Have Bugs (bugs means errors in the code)	99
13.7.4 4. The Devil Is In The Details I'm Hiding For Simplicity.	99
13.8 Welcome to the Dark Side	100
13.9 Sum Up: Bitcoin Coordinates Value, Ethereum Coordinates Value And Computation	100
13.10 Story Time...	100
13.11 But Ethereum Started with Proof-of-Work. Then Something Changed...	100
14 Chapter 14: When Consensus Splits - The Nature of Agreement	103
14.1 The Fundamental Question	103
14.2 Story 1: The Ethereum Fork - Code Is Law vs. Protect The Ecosystem	104
14.2.1 The DAO: A \$150 Million Experiment	104
14.2.2 The Hack: \$50 Million Stolen	104
14.2.3 The Dilemma: Two Incompatible Philosophies	105
14.2.4 The Vote: 85% vs. 15%	105
14.2.5 The Split: Two Etereums	106
14.2.6 Be Careful, The Past Can Haunt You	106
14.3 Story 2: Bitcoin vs. Bitcoin Cash - The Block Size War	107
14.3.1 The Problem: Bitcoin Is Slow	107
14.3.2 The Debate: Bigger Blocks vs. Keep It Small (size matters)	107
14.3.3 The Fork: Bitcoin vs. Bitcoin Cash	107
14.4 The Pattern: Technology Enables, Humans Decide	108
14.4.1 1. Technology Enables The Fork	108
14.4.2 2. Humans Decide The Outcome	108
14.4.3 3. Blockchain Makes Disagreement Auditible	108
14.5 Forks Are Civil Wars, Kind Of	109
14.5.1 You Can't Force Global Consensus	109
14.5.2 History Of Disagreement Is Preserved	110
14.6 The Deeper Realization: From Two Societies To Many	110

14.7 Enough Division—Why Not Unite?	110
14.8 What This Means For Anyone	111
14.9 The Anti-Gaslight Machine, Upgraded	111
15 Chapter 15: A Society For You! A Society For Me! A Society For Everyone! 113	
15.1 The Blockchain Trilemma (like a dilemma but with three options)	113
15.1.1 Decentralization + Security = Slow (Bitcoin, Ethereum)	113
15.1.2 Scalability + Security = Centralized	114
15.1.3 Decentralization + Scalability = Chaos (Insecure)	114
15.2 Why Decentralized CDNs Are Inherently Slow	115
15.3 Do We Need Slow Global Consensus For Everything Though?	115
15.4 Layer 2 Solutions: Mini-Societies Within A Mega-Society	115
15.4.1 Examples of Layer 2s	116
15.4.2 The United States	116
15.4.3 A Corruption Example	116
15.4.4 Another Example	117
15.4.5 Another Another Example	117
15.5 The Temporary Bad News: Trade-Offs	117
15.5.1 Technical Complexity Is Real	117
15.5.2 Cognitive Load: Too Many Choices	117
15.5.3 Different Trust Assumptions And Software Accessibility	118
15.5.4 Industry Age Context	119
15.5.5 Fragmentation	119
15.6 Summing up	119
15.7 Security Is Shared Globally	119
15.8 The Last Downside, Privacy	120
16 Chapter 16: Zero-Knowledge Proofs - Proving Without Revealing 123	
16.1 What Is a Zero-Knowledge Proof?	123
16.2 A Simple And Classic Analogy: The Colorblind Friend	124
16.3 The Mathematical Breakthrough	124
16.4 The Three Properties of Zero-Knowledge Proofs	125
16.4.1 1. Completeness	125
16.4.2 2. Soundness	125
16.4.3 3. Zero-Knowledge	125
16.5 How Does This Help Blockchains?	126
16.6 Real-World Examples	126
16.6.1 Example 1: Private Transactions	126
16.6.2 Example 2: Proving You're Over 18	126
16.6.3 Example 3: Private Voting	126
16.6.4 Example 4: Proving Solvency Without Revealing Balances	127
16.7 Why This Matters	127
16.8 The Catch: Complexity and Performance	127
16.8.1 Complexity	128
16.8.2 Performance	128
16.9 The Future: Private Coordination at Scale	128
16.10 But We're Not There Yet	128
16.11 A Note on Quantum Computers	129

17 Chapter 17: What This All Means for Humanity	131
17.1 What Was Impossible Before	131
17.2 What Changed?	132
17.2.1 Mathematics	132
17.2.2 Incentives	132
17.2.3 Social Consensus	133
17.3 The Power Dynamics Shift	133
17.3.1 Before: Centralized Gatekeepers	133
17.3.2 After: Distributed Coordination	133
17.3.3 The Anti-Gaslight Machine	134
17.4 Real-World Implications	134
17.4.1 Money: Cannot Be Frozen, Censored, or Arbitrarily Inflated	134
17.4.2 Governance: Transparent Voting, Credible Neutrality, DAOs	135
17.4.3 Identity: Own Your Data, Portable Across Platforms	135
17.4.4 Coordination: Layer 2s = Mini-Societies	135
17.4.5 Privacy: Zero-Knowledge = Prove Without Revealing	135
17.5 Flami's Farewell: Leaving the Propaganda Zone	135
17.6 The Philosophical Core	136
17.6.1 Consensus Is Social, Not Technical	136
17.6.2 Value Is Consensual In Complex Worlds	136
17.6.3 Coordination Is Voluntary	136
17.6.4 Technology Enables, Humans Decide	137
17.7 The Invitation	137
17.8 The Core Message	138
17.9 We've Unlocked New Forms of Coordination	138
18 Chapter 18: The Devil Is In The Details - A Reality Check	141
18.1 Part 1: Is Bitcoin Actually “Money”?	142
18.1.1 What I Told You	142
18.1.2 The Reality Check	142
18.1.3 Additional properties Bitcoin DOESN'T fully have (yet):	143
18.1.4 The Balanced Truth	143
18.2 Part 2: The Centralization Problem	144
18.2.1 What I Told You	144
18.2.2 The Reality Check	144
18.2.3 Mining concentration:	144
18.2.4 Geographic concentration:	144
18.2.5 Wealth concentration:	145
18.2.6 Ethereum's staking concentration:	145
18.2.7 Development concentration:	146
18.2.8 The Balanced Truth	147
18.3 Part 3: The Bitcoin Energy Problem	147
18.3.1 What I Told You	147
18.3.2 The Reality Check	147
18.4 Part 4: The Exaggerations	149
18.4.1 “Prove ownership of digital assets without central registry”:	149
18.4.2 “Coordinate without trusted intermediaries”:	149
18.4.3 “Physically impossible things are now possible”:	150

18.5 Part 5: Smart Contracts Aren't That Smart Nor Unstoppable	150
18.5.1 What I Told You	150
18.5.2 The Reality Check	150
18.6 Part 6: Layer 2s Don't Solve Everything	151
18.6.1 What I Told You	151
18.6.2 The Reality Check	152
18.7 Part 7: The No Trust Myth	153
18.7.1 What I Told You	153
18.7.2 The Reality Check	153
18.8 Part 8: The Use Case Problem	156
18.8.1 The Elephant in the Room	156
18.9 Part 9: The Ideology Problem	157
18.9.1 What I Told You	157
18.9.2 The Reality Check	157
18.10 Part 10: So What Should You Do?	158
18.11 Wrapping Up The Breaking Ball	159
18.11.1 I like trains.	159
18.11.2 The Balanced Truth	159
19 Chapter 19: From Bit to Bitcoin - Wrap-Up	163
19.1 The Journey	163
19.1.1 Part 1: Foundations - What Is Information?	163
19.1.2 Part 2: Trust & Cryptography - Who Can You Trust?	164
19.1.3 Part 3: Consensus Networks - The Breakthrough	164
19.1.4 Part 4: Evolution & Implications - What This All Enables	165
19.2 Recap: The "Weird" Terms (They Should Feel More Natural Now)	165
19.2.1 Bit	166
19.2.2 Logos	166
19.2.3 Algorithm	166
19.2.4 Protocol	166
19.2.5 Hash	166
19.2.6 Asymmetric Cryptography	166
19.2.7 Consensus	166
19.2.8 Proof-of-Work	166
19.2.9 Blockchain	167
19.2.10 Decentralized Datasync Technology and CDNs	167
19.2.11 Database and network	167
19.2.12 Smart Contracts	167
19.2.13 Forks	167
19.2.14 Layer 2	167
19.2.15 Zero-Knowledge	167
19.3 What You Now Understand	168
19.4 You Can Now...	168
19.5 The Meta-Insight	169
19.6 From Bit to Bitcoin	169
20 Chapter 20: Epilogue	171
20.1 Why I Wrote This	171

20.1.1	The Emotional Barrier	172
20.1.2	The Book's Goal	172
20.2	The Iron Law of Oligarchy - Why Education Matters	173
20.2.1	The Oligarchy of Nerds	173
20.3	Author Permission - This Book Is Free Forever	174
20.4	Why This Matters Beyond Technology	174
20.5	From Bit to Bitcoin	175

0

Chapter 0: Welcome to the Journey

0.1 What This Book Is

This is a book about understanding—not hype.

You've heard the buzzwords: Bitcoin, blockchain, cryptocurrency, Web3, NFTs, smart contracts. You've seen the headlines: fortunes made and lost, revolutionary promises, apocalyptic warnings.

But what is this technology, really?

This book will answer that question from first principles. We'll start with the most basic building block—the bit—and build up, step by step, until you understand how Bitcoin works, what blockchains actually do, and why this technology matters.

You don't need a computer science degree. You don't need to be good at math. You just need curiosity.

0.2 What You'll Learn

Part 1: Foundations (Chapters 1-8) The technical basics: bits, information theory, algorithms, cryptography, digital signatures, hash functions. These are the tools that make everything else possible.

Part 2: Bitcoin (Chapters 9-12) How Bitcoin works: money as synchronized bits, Proof-of-Work, consensus, the blockchain data structure. You'll understand the breakthrough that started it all.

Part 3: Beyond Bitcoin (Chapters 13-16) Ethereum and smart contracts, Layer 2 scaling solutions, Zero-Knowledge Proofs. The evolution of blockchain technology and what new capabilities it enables.

Part 4: Philosophy & Reality (Chapters 17-19) What this all means for humanity—the promises, the trade-offs, the philosophical implications. And most importantly, the reality check.

0.3 A Warning About Propaganda

This book, until Chapter 17, has been written with a propagandistic tone.

Not all statements you'll read are 100% true—the devil is in the details, and the devil appears in Chapter 18.

Why propaganda? Because revolutionary technologies are hard to introduce. If I led with all the problems, limitations, and failures, you'd stop reading on page 3.

So I've taken the role of a "crypto" enthusiast to keep you engaged. The **technical details**—bits, algorithms, cryptography, consensus mechanisms—are accurate, and you can trust those. But the **social implications**—revolutionary potential, adoption timelines, societal impact—are simplified, sometimes exaggerated, to maintain momentum.

Chapter 18 will demolish this propaganda with a comprehensive reality check.



Figure 1: Breaking Ball

You'll get the dream first. Then you'll get the reality. And by understanding both, you'll be able to form your own informed opinion.

0.4 Why Learn This?

Still, this technology is revolutionary—it enables humans to do things that were literally **physically impossible** before. It's like when cars were invented.

But due to the intrinsic nature of revolutionary technologies, if you don't simplify their introduction to someone who's never heard of them, it makes it way harder to start understanding them. Therefore, **welcome to the propaganda zone of this book**. Enjoy it, but keep in mind that it's not 100% accurate.

You'll see our friend Flami () when we leave the propaganda machine.

However, you can already read more reasons on why learning all this in Chapter 20. You will not understand a bit of that chapter without reading all the previous ones, but some main section can be understood now and they might even motivate you to go through the rest of this boring at certain parts book.

0.5 A Note on Tone

If you're a serious reader and the flamingo and my slightly informal tone bother you, I'm sorry. Most of the real nerds who know this technology deeply are like this.

Also, I'm 22, born in an era where nothing makes sense, where values are constantly blurred or destroyed, so we enjoy absurdity—like this flamingo: .

I hope the cultural differences of our generational gap do not play a crucial role in you reading and understanding the actual value encoded in this book.

0.6 A Note on Data Accuracy and AI Usage

The author has multiple projects at the same time, therefore, I tried to offload all the work that this books requires to AI. Some data has been searched and verified by AI, not all numbers presented might be accurate, yet, the concepts, dynamics and ways to understand the technology are.

Most of this book is AI written yet the metaphores, analogies and overall structure came from the author's brain.

0.7 The Promise

By the end of this book, you will understand:

- How Bitcoin actually works (not just what people say about it)
- What makes blockchain technology different from regular databases
- Why smart contracts are powerful (and limited)
- What Zero-Knowledge Proofs enable
- The honest trade-offs, limitations, and failures
- Whether this technology is right for specific use cases

- How to think critically about blockchain claims

You won't become a blockchain engineer, but you'll understand enough to cut through the hype and the panic, to separate signal from noise, and to make informed decisions about this technology.

0.8 Let's Begin

Without further ado, let's dive in!

Start with Chapter 1: The Bit—the simplest unit of information, and the foundation of everything digital.

This book is free because knowledge should be accessible. If you find value in it, share it with others please.

1

Chapter 1: The Bit

Everything starts with a choice: on or off, yes or no, 0 or 1.

Stand up and walk to the nearest light switch. Go ahead, I'll wait.

Now flip it. On. Off. On. Off.

Congratulations. You just performed the most fundamental operation in all of computing. You created a **bit**.

1.1 The Simplest Possible Information

A bit is the smallest unit of information that can exist. It's a choice between two states:

- **On or Off**
- **Yes or No**
- **True or False**
- **1 or 0**

That's it. Nothing simpler exists.

Think about it: you can't have "half on" or "kind of yes." A light switch is either up or down. A door is either open or closed. A coin shows either heads or tails. Two states. One choice.

Notice that, even if we get as philosophical as we can, things either exist or don't exist. To be or not to be. Two states. One choice. "Half to be" or "half not to be" doesn't make logical sense.

This simplicity is its power. This simplicity lays out the foundations of logic and modern computer communication systems.

1.2 From Light Switches to Computers

Your computer—the one you're likely reading this on right now—is made of billions of tiny switches. Not physical light switches you can flip with your finger, but microscopic electronic switches called **transistors** that can be turned on and off millions of times per second.

Each transistor holds one bit: on (1) or off (0).

Right now, as you read this sentence, billions of transistors inside your device are switching on and off in precise patterns. Some are holding the letters of this text. Others are tracking where your eyes are on the screen. Still others are keeping time, managing memory, rendering colors.

All of it—every webpage, every photo, every video, every song—is just patterns of bits. Billions of tiny switches, arranged just right.

1.3 Binary: The Language of Two

We call this system **binary** because it's based on two states (bi = two).

In our everyday life, we count using ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. We probably do this because we have ten fingers. When we run out of digits, we add another column and start over: 10, 11, 12...

Computers count using only two digits: 0 and 1. When they run out, they add another column. Let's see this side by side:

```
Human: 0
1
2
3
4
5
6
7
8
9
We're out of symbols! So we start over: put a 1, and add a 0 behind -> 10
10
11
12...
```

```
Binary: 0
1
We're out of symbols! So we start over: put a 1, and add a 0 behind -> 10
10
11
We're out again! Start over with 1, add 00 behind -> 100
100
101
110
111
1000...
```

It looks strange at first, but it's the same idea—just with two symbols instead of ten.

1.4 Eight Switches = One Byte

Let's make this concrete.

Imagine you have eight light switches in a row, or eight bits in a row:

[OFF] [OFF] [OFF] [OFF] [OFF] [OFF] [OFF] [OFF]

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Each switch can be either off (0) or on (1). That gives us **256 different possible patterns** ($2^8 = 256$). If you're curious about how to calculate the combinations with math, I encourage you to research and be curious. For now, those details won't be necessary.

We call eight bits grouped together a **byte**.

One byte can represent:

- Any number from 0 to 255
- One letter of the alphabet (A-Z, there are only 26 of them—we have 256 possible patterns to work with!)
- One character of punctuation
- One instruction for the computer to execute
- One pixel's color value
- A thousand other things

Here are some examples of what our eight switches might mean:

[OFF] [OFF] [OFF] [OFF] [OFF] [OFF] [OFF] [OFF]	-> 0
[OFF] [OFF] [OFF] [OFF] [OFF] [OFF] [ON]	-> 1
[OFF] [ON] [OFF] [OFF] [OFF] [OFF] [OFF]	-> 64
[ON] [OFF] [OFF] [ON] [OFF] [OFF] [OFF]	-> 144
[ON] [ON] [ON] [ON] [ON] [ON]	-> 255

The pattern of switches determines the value. Change one switch, change the meaning.

1.5 All Complexity Builds From This

Here's what might seem impossible: everything digital you've ever experienced—every game, every movie, every song, every website, every photo, every message—is built entirely from patterns of bits encoding the meaning we gave them.

Your favorite song? A very long sequence of 0s and 1s that, when interpreted correctly, recreates sound waves.

A photo of your family? Millions of bits encoding the color and brightness of each tiny pixel.

This sentence you're reading? Each letter is a specific 8-bit pattern your computer knows how to display.

The Bitcoin blockchain? A massive shared file made of bits, following precise rules about which patterns are valid.

Everything is bits.

The magic isn't in the bits themselves. The magic is in how we **interpret** them—how we assign **meaning** to patterns.

And that brings us to the next big idea: if bits are just patterns, who decides what those patterns mean? How does 01001000 become the letter "H"? How do we agree that a certain sequence of millions of bits represents a photograph and not random noise? How do we agree on which transistors will represent the bits that ultimately form a bit-coin?

The answer is profound and simple: **humans decide**. We make up rules, agree on standards, and build systems that follow those rules.

That's what the next chapter is about.

Languages like English work the same way—symbols in the shape of specific sounds, put together, creating what would otherwise be random noise but which we give meaning to by agreement. Bits are just a more fundamental level of the same idea, one that is “easy” to represent with physical objects.

1.6 Under the Hood: How Computers Store a Bit, Like Physically

A bit isn’t really stored as a “0” or “1”—those are just symbols we use. Physically, a bit is stored as:

In a transistor (modern computers): - A tiny switch made of silicon - “0” = no electrical charge flowing in the silicon - “1” = electrical charge flowing in the silicon - Billions fit on a chip smaller than your fingernail

In memory (RAM): - Tiny “batteries” that hold or release charge.

On a hard drive: - Magnetic regions that point north or south

The physical details differ, but the concept remains: **two distinguishable states** that we call 0 and 1.

We could, theoretically, make a computer out of doors and strings. Door open means 1 and door closed means 0. Pulling a string could turn a door on or off. It would be slow and impractical, but the principle is the same.

We could make Bitcoin with doors—door-coin if you like. But silicon transistors, as of now, are the best we have, in the sense that they’re just way faster and smaller.

Key Insight: All complexity in computing, all digital technology, all of cryptocurrency and blockchain—everything builds from this one simple foundation: the bit. A choice between two states. On or off. Yes or no. 0 or 1.

Master this idea, and everything else becomes understandable. It logically follows.

Next, we’ll see how these meaningless patterns become meaningful information.

2

Chapter 2: Logos - Mapping Meaning to Numbers

Patterns mean nothing until we decide they mean something.

We ended the last chapter with a question: If bits are just patterns of 0s and 1s, who decides what those patterns mean?

The answer is both profound and simple: **we do**. Humans. We made it all up. And that's not a weakness—it's the entire point.

2.1 The Meaning Problem

Imagine I hand you this sequence:

01001000 | 01000101 | 01001100 | 01001100 | 01001111

What is it? To your computer, it's just five groups of eight switches—some on, some off. The pattern exists, but it has no inherent meaning. It's not “secretly” anything. It's just... a pattern.

But if I tell you we're using a system called **ASCII** (American Standard Code for Information Interchange), suddenly those patterns become:

01001000 → H
01000101 → E
01001100 → L
01001100 → L
01001111 → O

HELLO.

The bits didn't change. The pattern didn't change. What changed was that we agreed on a **mapping**: a table that says “this number means this letter.”

This is the idea of **logos**—meaning assigned to form. The word, the pattern, the symbol that carries significance because we collectively agree it does.

2.2 The ASCII Table: A Social Agreement

In the 1960s, a group of people sat in a room and made a decision. They decided:

- The number 65 would represent the letter ‘A’
- The number 66 would represent the letter ‘B’
- The number 67 would represent the letter ‘C’
- ...and so on

They could have chosen any numbers. They could have made ‘A’ equal to 200, or 7, or 42, or 43, 69 even, if they woke up horny. It was arbitrary. But once they agreed and wrote it down, it became a **standard**.

ASCII was first standardized in **1963**, with revised versions following in 1967 and 1968.

Here’s a tiny slice of the ASCII table:

Decimal (human counting)	Binary	Character
65	01000001	A
66	01000010	B
67	01000011	C
...		
72	01001000	H
...		
90	01011010	Z

Now, when your computer sees 01000001, it knows to display the letter ‘A’ on your screen—not because there’s something magical about that pattern, but because someone decided it, everyone agreed, and we all use the same table.

This is both a standard and a protocol. A standard is an agreed-upon specification (the ASCII table itself), while a protocol is a set of rules for communication (how computers use that table to exchange text). In practice, these terms are often used interchangeably when talking about shared agreements. Just like we agreed that the sound “cat” refers to a small furry animal, we agreed that 01000001 refers to the letter ‘A’.

2.3 My Binary BRO

Let’s become a bro, as the teenagers say, and write it in binary: **BRO**

First, look up each letter in the ASCII table:

Letter	Decimal	Binary
B	66	01000010
R	82	01010010
O	79	01001111

So “BRO” in binary is:

01000010 01010010 01001111

That’s it. That’s the word, represented as 24 switches (3 letters times 8 bits each). Change one switch, and it’s a different word. Change the table we’re using, and the same pattern could mean something completely different.

Here's where it gets interesting: If someone in the '60s had decided that 66 meant 'B', 82 meant 'R', but 79 meant 'A' instead of 'O', then our bits 01000010 01010010 01001111 would spell "BRA" for us today instead of "BRO".

Imagine your computer says "I really love my bro," but someone interprets it with a different table, and it becomes "I really love my bra." Some might blush. This is the power of agreed meaning, and the importance of standards and protocols for understanding and coordinating communication without misunderstandings.

Try it yourself: If you're curious and want a mental exercise, look up the ASCII table online and encode your name. Each letter is a number, each number is a pattern of bits. Whatever YOU mean is encodable.

2.4 Beyond Letters: Everything is Numbers

The same trick works for everything digital:

Images: You break the image into tiny squares called pixels, each with a color. Each color is a number (e.g., Red=255, Green=128, Blue=64), and you store millions of these numbers in a file. When you open the file, the computer reconstructs the image by reading those numbers back.

Sound: You measure the air pressure thousands of times per second, and each measurement becomes a number. Store those numbers in sequence, play them back, and your speakers recreate the sound wave.

Video: A video is just many images (called frames) displayed in rapid sequence, plus a soundtrack. All numbers, all bits.

Bitcoin: Account balances are numbers. Transaction amounts are numbers. Cryptographic signatures are numbers. The entire blockchain is just a very long sequence of bits following specific rules.

Notice the pattern: **Everything becomes a number. Every number becomes bits. We assign meaning to those bits.**

2.5 The Deep Insight: Information is Agreement

Here's the key realization: **Information doesn't exist "out there" in the universe. We create it by agreeing on meaning.**

The sequence 01001000 is not "the letter H." It's just a pattern. But when billions of people use computers that follow the ASCII standard, it *becomes* the letter H for practical purposes. The meaning emerges from collective agreement.

This might sound abstract, but it's crucial for understanding Bitcoin, because Bitcoin is the same idea taken further:

- Gold is valuable because we agree it's valuable (and because it's useful sometimes, rare, portable, etc.—but you get the first idea for now: **agreement is a source of value**)
- Dollars are valuable because we agree they're valuable (and the government enforces it)
- **Bitcoin is valuable because participants agree it's valuable—and the agreement is encoded in software that everyone runs**

The bits themselves? Meaningless. But the *coordinated interpretation* of those bits? That creates money, contracts, organizations, entire economies.

2.6 Languages Work the Same Way

Think about the word “cat.”

C-A-T

It’s just three sounds we make with our mouths, or three symbols we write on paper. There’s nothing inherently “catty” about the sound “cat.” In Spanish, it’s “gato.” In Japanese, it’s “neko” (ねこ). Same animal, different sounds, different symbols.

But within English-speaking communities, we all agree: the sound “cat” refers to that small furry animal. Change the agreement, and the same sounds could mean something else entirely. In French, “chat” (pronounced differently) means cat.

Bits work the same way, just at a more fundamental level. We’re assigning meaning to the shapes of transistor states instead of the shapes of mouth sounds or pen strokes.

Language is meaning mapped to sound patterns. Writing is meaning mapped to visual patterns. Computing is meaning mapped to electrical patterns.

And Bitcoin? Bitcoin is **value mapped to bit patterns**, with the mapping enforced not by governments or banks, but by math and code representing aligned incentives which its community agrees are valuable.

2.7 Why This Matters for Bitcoin

You might be wondering: why are we talking about ASCII tables in a book about Bitcoin?

Because Bitcoin is built on the same foundation. It’s bits following rules, where the rules are socially agreed upon.

- A Bitcoin private key? A 256-bit number.
- A Bitcoin transaction? A specific pattern of bits with a particular structure.
- The blockchain? A sequence of these patterns, linked together with cryptographic hashes. Hashes? Are we talking about drugs? Don’t worry, we’ll get there, and it will be intuitive and logical.

The Bitcoin protocol is like the ASCII table, but for money: - “This pattern of bits represents 1 BTC” - “This pattern proves Alice owns it” - “This pattern transfers it from Alice to Bob”

None of it is “real” in the sense that gold is real. But it’s real in the sense that language is real: it exists because we collectively act as if it does, following shared rules.

Actually, yes, some part *is* real: millions of transistors in very specific computers around the world, synchronized in a very specific way. THAT IS BITCOIN.

And that’s powerful. Because unlike gold (heavy, hard to divide, hard to transport) or government money (controlled by central authorities), Bitcoin is: - Pure information (move it at the speed

of light) - Perfectly divisible (down to 0.00000001 BTC) - Globally accessible (anyone with internet can participate) - **And the rules are transparent, auditable, and enforced by coded mathematics and algorithms representing certain incentives**

But we're getting ahead of ourselves. We still need to understand how computers actually *do* things with these bits. How do they follow the rules? How do they take 01001000 and turn it into the letter 'H' on your screen?

That's what the next chapter is about: **algorithms**—the instruction manuals that make computers useful.

Key Insight: Information is meaning we assign to patterns. Computers don't "understand" anything—they follow agreed-upon mappings (like ASCII) that translate bits into things humans recognize. All digital information, including money, is built on this foundation of social agreement encoded in protocols.

Next, we'll see how computers actually *process* this information. How do they take rules like "01001000 = H" and execute them billions of times per second? That's the power of **algorithms**—and they're likely simpler than what you might expect.

3

Chapter 3: Algorithms - Following Rules

Computers don't think. They follow recipes, very fast, very precisely.

We've established that bits are just patterns, and we assign meaning to those patterns through agreed-upon mappings like ASCII. But how do computers actually *do* anything with this information?

The answer: **algorithms**. Fancy word, simple concept. An algorithm is just a set of instructions—a recipe for solving a problem or completing a task.

3.1 The Recipe Analogy

Think about making a peanut butter and jelly sandwich. Here's the algorithm:

1. Get two slices of bread
2. Get peanut butter jar
3. Get jelly jar
4. Get a knife
5. Open peanut butter jar
6. Dip knife in peanut butter
7. Spread peanut butter on one slice of bread
8. Clean knife
9. Open jelly jar
10. Dip knife in jelly
11. Spread jelly on the other slice of bread
12. Put the two slices together
13. Done!

That's an algorithm—a sequence of steps that, if followed exactly, produces the desired result.

The key property: If you follow the same steps with the same ingredients, you get the same sandwich every time. This is called being **deterministic**: same input leads to the same output, always. Think of input as “initial information” or “initial symbols,” and think of output as “final information” or “final symbols.”

Computers work the same way. Give a computer an algorithm and some input data (from now on, if you read the word “data” you can also think of it as “information”), and it will execute the steps

exactly as written, producing the same output every single time. And if not? Well, you've been hacked, or you should contact your computer's manufacturer—they messed up some transistor or cable or other physical component.

3.2 A Simple Algorithm: Binary to Letters

Let's write an algorithm that takes binary data and converts it to text using the ASCII table we learned about in Chapter 2.

Input: A string of bits (e.g., 01001000 01001001)

Output: Human-readable text (e.g., "HI")

Algorithm:

Step 1: Split the input into groups of 8 bits
 01001000 | 01001001

Step 2: Convert each group from binary to decimal
 01001000 = 72
 01001001 = 73

Step 3: Look up each decimal number in the ASCII table
 72 = H
 73 = I

Step 4: Put the letters together
 Output: "HI"

That's it. Four steps. Follow them precisely, and 01001000 01001001 becomes "HI" every time.

Your computer does this billions of times per second. When you open a text file, load a webpage, or read this sentence, your computer is running algorithms that convert bits into pixels, sounds, letters, and images.

3.3 Computers Don't “Think”

Here's something crucial to understand: **Computers don't think. They don't understand. They just follow instructions.**

When your computer displays the letter 'H', it doesn't "know" what 'H' means. It doesn't understand the concept of letters or language. It simply followed this algorithm:

```
IF bits = 01001000
THEN display pixel pattern #72 from font file
```

That's all. No comprehension, no intelligence, no awareness. Just: see this pattern, do this action.

This might seem disappointing, but it's actually profound. By following simple rules very fast, computers can do things that *look* intelligent:

- Translate languages (following grammar rules + dictionary lookups)

- Play chess (following move-evaluation rules)
- Recognize faces (following pattern-matching rules)
- Route your emails (following network protocol rules)
- **Process digital money transactions (following validation rules)**

It's all algorithms. All instructions. All deterministic.

3.4 The Power of Determinism

Deterministic means: given the same input, you always get the same output.

This property is critical for computers working together. Imagine if your computer and my computer could look at the same bits and get *different* results—communication would be impossible, and coordination would break down entirely.

Bitcoin relies on this property completely. Thousands of computers around the world run the same algorithms on the same data, and they all arrive at the same conclusions about what's valid and what's not.

This is why Bitcoin works. Not because of magic, but because deterministic algorithms guarantee that everyone following the same rules ends up with the same “truth”—predictable, verifiable information.

What is a miner? A block? A transaction? We will get there, don't worry. For now, just understand that all of these concepts are built on top of algorithms that computers follow precisely.

3.5 Algorithms Can Be Simple or Complex

Some algorithms are trivial:

```
Algorithm: Add two numbers
Input: 5, 3
Step 1: Add the numbers together
Output: 8
```

Others are incredibly complex:

```
Algorithm: Render a 3D video game frame
Input: Player position, world data, lighting rules, physics...
Step 1: Calculate which objects are visible
Step 2: Apply lighting and shadows
Step 3: Apply textures to surfaces
Step 4: Calculate reflections
Step 5: Apply motion blur
Step 6: Convert 3D coordinates to 2D pixels
... (hundreds more steps)
Output: One frame of the game (1/60th of a second)
```

But both are the same fundamental idea: a sequence of instructions that transforms input into output.

Bitcoin uses both kinds—simple algorithms and complex ones. But all of it is deterministic. All of it is just following rules.

3.6 Computers Follow Recipes Perfectly (and Stupidly)

Here's a classic example that shows both the power and limitation of algorithms.

Imagine you give a computer this algorithm:

Algorithm: Make a sandwich

Step 1: Put peanut butter on bread

A human would understand: get bread, open the peanut butter jar, spread it, etc.

But a computer would fail immediately. **“Put”?** **What does that mean?** **“Peanut butter”?** **Where is it?** **“On”?** **What’s the position?**

Computers need *every single step* spelled out explicitly:

```
Step 1: Locate object labeled "bread" in coordinate system
Step 2: Locate object labeled "jar_peanut_butter"
Step 3: Rotate jar lid counterclockwise 720 degrees
Step 4: Insert knife into jar
Step 5: Move knife to coordinate X, Y, Z
... (hundreds of micro-steps)
```

This is why programming is hard—you must think like a computer: break everything down into the smallest possible steps, assume nothing, define everything.

But once you do: the computer executes those steps flawlessly, billions of times, without ever getting tired or making a mistake.

3.7 Why This Matters for Bitcoin

Bitcoin is a collection of algorithms.

These algorithms define things like: - How to verify that someone really owns the money they're trying to spend - How to make sure the same money isn't spent twice - How to agree on the order of transactions - How to reward people who help secure the network

Computers all over the world run these same algorithms. They all follow the same rules. Given the same information, they all arrive at the same conclusion about what's valid.

No trust needed. Just math.

Well, not exactly “no trust.” You still trust: - The algorithms are correctly implemented - The majority of participants are following the rules - Your computer hardware isn't lying to you

But you don't need to trust any single person or institution. You can verify everything yourself by running the algorithms.

This is what people mean by “trustless” (though “trust-minimized” is more accurate). You're eventually trusting math and code running an agreed consensus, not bankers and governments, which might put the money where you told them... or not.

3.8 The Transition to Protocols

Algorithms tell individual computers what to do. But what about computers talking to *each other*?

That requires **protocols**—agreed-upon rules for communication. And that’s exactly what we’ll explore in the next chapter.

Protocols, like... The Internet Protocol! Voila, that word we all use but little understand: Internet.

Because Bitcoin isn’t just one computer running algorithms. It’s thousands of computers coordinating across the internet, all running the same algorithms, all speaking the same protocol or standard, all converging on the same “truth” (ordered information interpreted with the same meaning).

And *that’s* when it gets really interesting.

Key Insight: Algorithms are instruction sets that computers follow deterministically. Same input leads to the same output, always. This determinism is what allows thousands of computers to independently verify the same information and reach the same conclusions without trusting each other. Computers don’t “understand” anything—they just follow rules perfectly, billions of times per second.

Next, we’ll see how computers coordinate across networks. How does your computer talk to a server in another country? How do thousands of Bitcoin nodes stay synchronized? That’s the power of **protocols**—and they’re simpler than the word sounds.

Maybe you didn’t expect to leave this book with an intuitive understanding of the famous Internet. I hope it is as fascinating for you as it was for me when I first learned this years ago.

4

Chapter 4: Protocols - Computers Talking

Protocols are social agreements between machines.

We've covered how computers store information (bits), how they assign meaning to patterns (ASCII-like standards), and how they process that information (algorithms). But there's one crucial piece missing:

How do computers talk to each other?

When you load a webpage, send an email, or make a video call, your computer is communicating with other computers—often thousands of miles away. How does that work? How do they understand each other?

The answer: **protocols**. Agreed-upon rules for communication.

4.1 The Allergy Protocol

Let's start with a simple example. Imagine Alice wants to tell Bob's computer a secret, but she's worried someone else might be pretending to be Bob. She needs a way to verify his identity. By the way, Alice is Bob's doctor.

Here's the thing: Bob is allergic to bananas. This is medical history—private information that only Bob knows. Alice can use this to verify she's really talking to Bob, without Bob having to reveal his medical history to anyone listening on the network.

So she invents a protocol:

Alice's Protocol:

Rule: Anyone claiming to be Bob must correctly answer "What are you allergic to?" Only the real Bob knows the answer.

Now watch it in action:

Alice: "What are you allergic to?"

Bob: "Bananas."

Alice: "Okay, you're Bob. Here's the secret: Melisa is dating Bryan."

This is a protocol. A set of rules both parties follow to communicate successfully. Once identity is verified with information that only Bob knows, Alice can safely share the secret, and communication begins.

What if someone else tries?

```
Alice: "What are you allergic to?"
Carol (pretending to be Bob): "Uh... peanuts?"
Alice: "Wrong. You're not Bob. Go away."
```

The protocol works because both parties know the rules in advance (asking for allergies in this case), following the rules proves identity (Bob knows private information), and not following the rules means rejection.

This is a network! Alice and Bob coordinating through agreed-upon rules. Scale this up to billions of computers, and you have the Internet.

Note: Computers do this with more complex stuff than allergies or medical data. They do it with **cryptography**—mathematical functions that prove identity without revealing secrets. We'll dive deep into this in the next part of the book. Do not worry, it will still be intuitive.

4.2 The Internet is Just Protocols

When you type “instagram.com” into your browser, here’s what actually happens (simplified):

```
Your computer: "Hey, give me instagram.com"
Instagram server: "Okay, here's the webpage data"
Your computer: "Cool. By the way, I'm Bob (here's my login info that proves it's me)"
Instagram: "Oh hey Bob! Here's your personalized feed. Nice cat pictures btw"
```

This conversation follows a protocol called **HTTP** (Hypertext Transfer Protocol). Every web browser and every web server speaks this protocol. They all agree on the format:

REQUEST format:
 GET /home HTTP/1.1
 Host: instagram.com
 Cookie: session_id=abc123

RESPONSE format:
 HTTP/1.1 200 OK
 Content-Type: text/html
 [webpage data here...]

This weird format, instead of asking about allergies, is asking stuff like: Hey man, what is the size of your screen? You: this big. Now the server knows how big the images are that it has to send you, for example. And way more questions are asked, sure, but this is the basic dynamic you should understand.

Your browser doesn’t need to “know” Instagram—or any certain website or app—exists. Instagram doesn’t need to “know” about your specific browser or computer. They just both follow the HTTP protocol, so they can communicate without ever having met.

This is the power of protocols: Strangers can coordinate without ever meeting, as long as they follow the same rules.

4.3 Protocols are Everywhere

Think about human communication. We have protocols too:

The English Language Protocol: - Words have agreed-upon meanings (remember ASCII?) - Grammar rules structure sentences - Context helps disambiguate - If both people follow these rules, communication works

The Phone Call Protocol: 1. Person A: "Hello?" 2. Person B: "Hi, is this [name]?" 3. Person A: "Yes, speaking." 4. Person B: [states purpose of call: "Bob is choking on a banana, please call an ambulance!"]

We don't think of these as "protocols" because they're so natural to us, but they are indeed protocols in a sense—agreed-upon rules that enable coordination.

Computers need protocols too, but they can't improvise like humans. They need **exact** specifications:

Email Protocol (SMTP - Simplified):

```
Step 1: Connect to mail server on port 25
Step 2: Say "HELO" to introduce yourself
Step 3: Say "MAIL FROM: sender@example.com"
Step 4: Say "RCPT TO: recipient@example.com"
Step 5: Say "DATA" and send the email content
Step 6: Say "QUIT" to close connection
```

Every email client and every email server follows these exact steps. That's why Gmail can send emails to Outlook, which can send to ProtonMail, which can send back to Gmail. **They all speak the same protocol.**

4.4 The Internet Protocol (IP)

The big one. The protocol that makes the Internet possible.

When you send data across the Internet, it gets broken into little chunks called **packets**. Each packet has: - **The data** (part of your message, like a cat picture you want to post on Instagram) - **Source address** (where it came from—your device) - **Destination address** (where it's going—Instagram's server)

Think of it like mailing a letter: the letter content is your data, the return address is your source IP (IP = Internet Protocol) address, and the recipient address is your destination IP address.

Routers along the way look at the destination address and forward the packet toward its destination. They don't need to know what's inside—they just follow the protocol: "Read destination address, forward to next hop."

Your computer's address? Something like 192.168.1.5 or 203.0.113.42.

Instagram's address? Something like 31.13.64.35.

Every device on the Internet has an address. The Internet Protocol (IP) is the set of rules for how to format packets and route them between addresses.

This is the Internet. Not a physical thing, not a cloud, not a magical ether—just billions of computers following the same protocol to send packets to each other. Sometimes via electricity in large or small cables, sometimes via electromagnetic waves, but always following the same rules.

4.5 Bitcoin is a Protocol Too

Now here's the connection: **Bitcoin is a protocol.**

Just like HTTP defines how web browsers and servers communicate, Bitcoin defines how nodes in the Bitcoin network communicate. A node is a fancy word for computer, or any device that runs on binary processing of information in a network.

The Bitcoin Protocol specifies stuff like: - How to format a transaction - How to broadcast it to the network - How to validate it - How to bundle transactions into blocks - How to agree on which block is next - How to reward miners - How to prevent double-spending

Every Bitcoin node runs software that follows this protocol. When a new transaction happens:

Node A: "Hey everyone, here's a new transaction: Alice → Bob, 0.5 BTC"

[broadcasts to all connected nodes, broadcast means like, sending it out to everyone]

Node B receives it:

- Step 1: Is the signature valid? (Check)
- Step 2: Does Alice have 0.5 BTC? (Check)
- Step 3: Has this been spent before? (Check)
- Step 4: Valid! → Store it, forward it to my peers

Node C receives it from Node B:

[Runs same validation]

Valid! → Store it, forward to my peers

[Transaction spreads across the network within seconds]

No central server. No one “in charge.” Just thousands of computers following the same protocol, independently validating the same rules, converging on the same truth.

4.6 Protocols Enable Trust Without Authority

Here's the profound realization:

With traditional systems, you need a trusted authority: banks validate your transactions, email providers (like Gmail or Outlook—not the same as the email protocol itself) deliver your messages and can read them, and governments issue your identity documents.

But with protocols, you can have **coordination without authority**: - The Internet works because everyone follows IP, not because someone “runs” the Internet - Email works because everyone follows SMTP, not because one company controls email - Bitcoin works because everyone follows the Bitcoin protocol, not because someone controls Bitcoin

The protocol is the authority. The rules are transparent, auditable, and enforced by math and code, not by institutions.

YOU decide which consensus to run with your code. Do I want 21 million maximum supply? More? Less? You state what you want by representing it in software, and if enough people agree—plus more engineering details we’ll share later—that becomes the standard, the protocol.

4.7 Why Protocols Matter for Bitcoin

Bitcoin solves the “who do you trust controlling, issuing, or storing money?” problem by replacing trust in institutions with trust in a protocol.

Instead of trusting a bank to keep accurate records, not freeze your account, not inflate the supply, and not censor your transactions—you trust the Bitcoin protocol (transparent rules), math (cryptography works), incentives (miners and nodes follow the rules because it’s profitable), and the majority (51%+ are honest, making attacks expensive or irrational).

This is the insight: Protocols can coordinate strangers at global scale without anyone being “in charge.”

The Internet demonstrated this for information. Bitcoin took inspiration from it and decided to interpret that information as money. But Bitcoin is not just one entity—it is all of the people, at the same time.

4.8 The Network Effect

Here’s why protocols become powerful:

Once enough people adopt a protocol, it becomes the standard. Email didn’t win because it was perfect—it won because everyone used it. The Internet Protocol didn’t win because it was optimal—it won because everyone adopted it.

Bitcoin is the same. As more people run Bitcoin nodes, accept Bitcoin payments, and hold Bitcoin, the network becomes more valuable—not because Bitcoin is technically “better” than alternatives, but because **more people follow the protocol**.

This is called the **network effect**: The value of a network grows exponentially with the number of participants.

- One phone = useless
- Two phones = one connection
- Ten phones = 45 possible connections
- One million phones = half a trillion possible connections

Bitcoin’s security comes from this. The more nodes, the harder to attack. The more miners, the more expensive to rewrite history. The more users, the more valuable the network.

We will later generalize the idea of a blockchain network, so you not only understand Bitcoin, but also leave this book with the ability to start understanding other blockchain network protocols as well (such as Ethereum and Solana, for example).

It is very important that you understand and learn to think about them in general, because **YOU CHOOSE WHICH PROTOCOL YOU RUN WITH YOUR COMPUTER**—and therefore, which information eventually and actually gets a bit more value thanks to your “belief,” your interpretation of those coordinated bits being something you want to use as money, for example.

4.9 Protocols are Living Standards

One last thing: Protocols can evolve, but it’s hard.

To change a protocol, you need **consensus** among all participants. Otherwise, you break compatibility.

Example: If Gmail suddenly decided to change how it sends emails, and Outlook didn’t update, emails between them would stop working.

This is why protocol changes are slow (needing widespread agreement), carefully coordinated (all participants must upgrade), and rare (too risky to change often).

Bitcoin is the same. Changes to the Bitcoin protocol require consensus across miners (a special type of node—computers which follow some extra rules), nodes, and users. This is intentional—it prevents anyone from arbitrarily changing the rules.

We’ll explore how this consensus works in much more detail later. For now, just understand: **Protocols are social agreements encoded in software, coordinating computers at global scale.**

Key Insight: Protocols are agreed-upon rules for communication. They enable strangers to coordinate without trusting any central authority. The Internet is a protocol (IP). Bitcoin is a protocol (the Bitcoin network). When millions follow the same protocol, you get emergent coordination—no one in charge, but everyone following the same rules, converging on the same truth.

Next, we enter **Part 2: Trust & Cryptography**. Because here’s the problem: If computers are talking across the Internet, anyone can listen. How do you send secrets over public channels? How do you make sure you are talking to someone you know? How do you prove who you are without revealing your password so no one can use it on your behalf? That’s what cryptography solves—and it’s one of the foundations of Bitcoin’s security.

5

Chapter 5: The Trust Problem

The internet is public by default. More open than your ex's legs.

We've established that computers communicate using protocols—agreed-upon rules that enable coordination across the globe. The Internet works because billions of devices follow the same rules to send packets of data to each other.

But here's the problem: **The Internet is a public network.**

If you visualize the computers connected to each other with lines, it looks like a net:

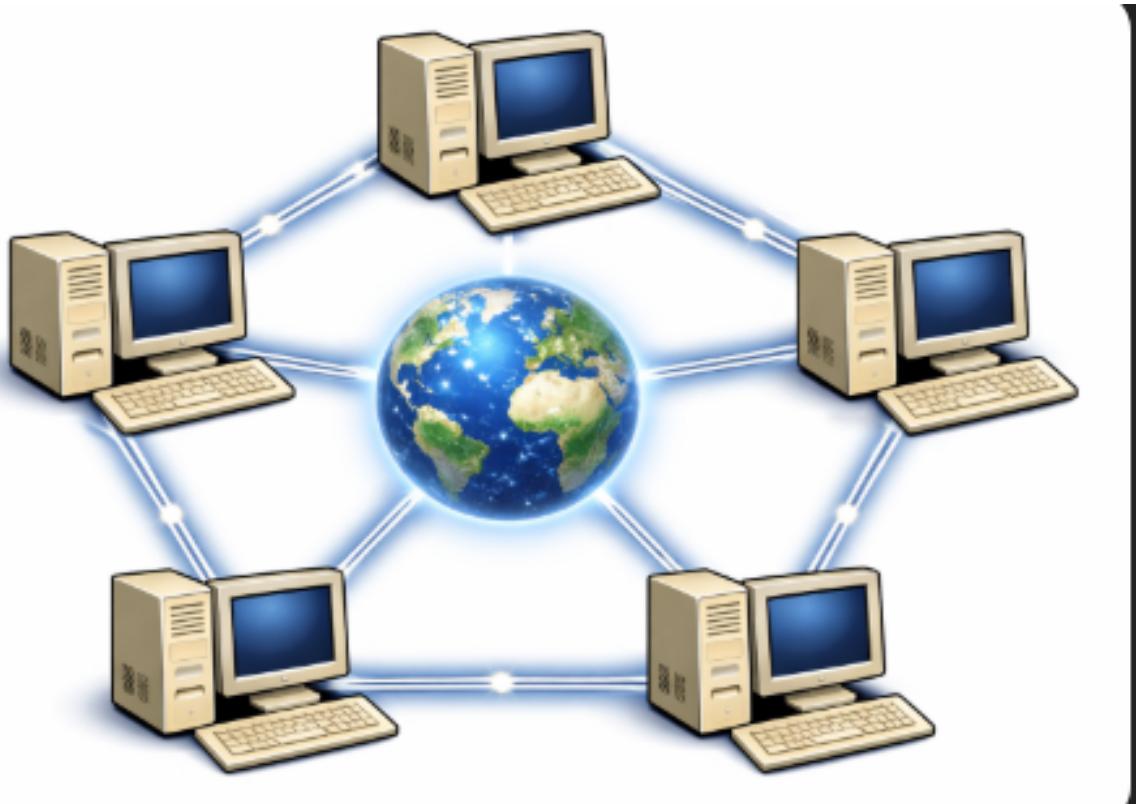


Figure 5.1: internet

When you send data across the Internet, it doesn't travel directly from your computer to the

destination. Instead, it hops through dozens of intermediate computers—routers, switches, servers—each one forwarding your packet closer to its destination.

Anyone along that path can read your data.

5.1 Alice, Bob, and Carol

Let's use a classic scenario from cryptography.

Alice wants to send Bob a secret message: "The password is: banana123"

She sends it across the Internet, and the message travels through: - Her local router - Her internet service provider (ISP) - Multiple backbone routers - Bob's ISP - Bob's local router - Finally, Bob's computer

At every single one of these stops, someone could be watching.

Enter Carol. She's an eavesdropper—maybe she works at the ISP, maybe she's running a router, or maybe she's just someone who figured out how to intercept network traffic (this is called **packet sniffing**).

```
Alice sends: "The password is: banana123"
  |
[Router 1] <- Carol is listening here!
  |
[Router 2]
  |
[Router 3] <- Carol could be listening here too!
  |
Bob receives: "The password is: banana123"
```

Carol sees everything. The password, the message, all of it. Alice and Bob have no privacy.

This is the trust problem: How do you send secrets over a public channel when anyone might be listening?

5.2 Historical Solutions

This isn't a new problem—humans have been trying to send secret messages for thousands of years.

Ancient methods included trusted messengers (who could be captured or bribed), sealed letters (which could be opened and resealed), and secret codes written on paper (which required both parties to have the code book).

World War II saw the famous Enigma machine, Germany's encryption device. Allied code-breakers spent years working to crack it, and once they did, they could read all German military communications. Wars were won and lost based on who could keep their messages secret.

The pattern: Throughout history, sending secrets has required trusting someone or something not to intercept them.

5.3 The Fundamental Question

Here's what makes the Internet different from all previous communication methods:

The Internet is designed to be open. The foundations of the Internet were laid with **ARPANET in 1969**, which connected trusted institutions—universities, research labs, government facilities. They shared academic papers and research data, and privacy wasn't the priority; open communication was. The modern “Internet” as we know it emerged through the 1980s and 1990s with the adoption of TCP/IP protocols.

But as the Internet grew and connected the whole world, a problem emerged: how do you send private data across a public network? Inside a building, dedicated cables work fine. But between cities, countries, continents? You need infrastructure—routers, cables, satellites—owned and operated by strangers.

It's not like a private phone line between two people. It's a global public network where data bounces through dozens of strangers' computers before reaching its destination.

This creates a paradox: - We need the Internet to be open (so we have a voice, therefore influence, everywhere) - But we need our messages to be private (so only the people we want can read them)

Can we have privacy in public?

For most of human history, the answer was: No. You can't send a secret in public without someone potentially reading it. You need private channels, trusted couriers, secure rooms.

But then mathematicians discovered something remarkable.

5.4 Why This Matters Today

Think about what you do online:

- **Banking:** You send passwords and account details. If Carol intercepts this, she steals your money.
- **Shopping:** You enter credit card numbers that travel across public networks.
- **Messages:** Medical records, legal documents, personal photos—all bouncing through strangers' computers.

The question we need to answer: How do Alice and Bob send secrets to each other when Carol is listening on the network?

The naive solution: “Just don't let Carol listen!”

But that's impossible. Carol could be: - An employee at your ISP - A government agency with access to routers - A hacker who compromised a network - The owner of a public WiFi network you're using - Anyone between you and your destination

You can't control who's watching. You can only control what they see.

5.5 The Solution: Encryption

You've probably seen this in your web browser: a little padlock icon next to the website name. It's usually green, and next to “https://”, instead of “http://”.

```
http://example.com  <- Not secure (Carol can read everything)
https://example.com <- Secure (Carol sees gibberish)
```

That little “s” at the end of “https” stands for “secure.” It means your connection is **encrypted**.

When you visit an HTTPS website:

- Your browser and the website establish an encrypted connection
- Everything you send looks like random noise to anyone watching
- Carol can see that you’re communicating, but not what you’re saying
- It’s like whispering in a language only you and the website understand

This is why online banking works. Not because the Internet Protocol is secure by default (it’s not), but because we encrypt the messages we send over it.

You can imagine it like two people talking in a language you don’t know, right in front of you. For you, that communication is encrypted. As an initial way of understanding it, computers do something similar but over the internet.

Remember ASCII, the standard everyone understands? Well, imagine your computer—because it can compute and assign meanings very, very fast—creates a new ASCII-like mapping with the destination computer, and then only they know what’s going on. Something like: “Hey, all Bs will be As, all As will be Bs and all Ys will be Zs”. Then the word BABY becomes ABAZ, and only they know. Later we’ll see how it actually works, but this first mental model will be helpful to you.

So how does this encryption actually work?

5.6 The Setup for What’s Next

We need to solve two related but different problems:

Problem 1: How to send secrets when everyone is watching - Alice wants to send Bob a private message - Carol is listening - The message must arrive intact and unread

Problem 2: How to prove identity - Alice receives an encrypted message - But how does she know it’s really from Bob? - What if Carol figured out the “language” and is impersonating Bob? - How do you prove identity across long distances when you can’t see each other?

Here’s the good news: **encryption solves both problems.** We’ll understand intuitively how later.

By the way, readers, encryption uses **math**. Both problems rely on the same fundamental breakthrough: **asymmetric cryptography**—one of the most important mathematical discoveries of the 20th century.

But before we get to that magic, we need to understand the simpler version first: **symmetric encryption**.

Because symmetric encryption sets up the problem that asymmetric encryption solves. And understanding the problem is half the battle.

Key Insight: The Internet is public by default. Anyone between you and your destination can potentially read your data. Throughout history, sending secrets required private channels or trusted couriers. But modern cryptography enables something that seems impossible: sending secrets in

public, where everyone can see the message but no one can read it. This is the foundation of digital privacy—and it's essential for Bitcoin.

Let's dive into the very basic concepts of encryption you need to feel safer and understand how Bitcoin works.

6

Chapter 6: Symmetric Encryption - Shared Secrets

Like a lock and a key—but both parties need the same key.

We've established the problem: Alice wants to send Bob a secret message, but Carol is listening on the network. How can they communicate privately?

The answer starts simple: **scramble the message so only Bob can unscramble it.**

This is called **encryption**, and the simplest form is called **symmetric encryption**—where both parties share the same secret.

6.1 The Caesar Cipher

Let's start with one of the oldest encryption methods: the Caesar cipher, named after Julius Caesar who used it to send military messages over **2,000 years ago** (Julius Caesar lived from 100 BCE to 44 BCE, making it approximately 2,050 years ago).

The idea is beautifully simple: **shift every letter by a fixed number.**

Example: Shift by 3

Original alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Shifted left by 1: B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
Shifted left by 2: C D E F G H I J K L M N O P Q R S T U V W X Y Z A B
Shifted left by 3: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Now encode a message:

Original alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Shifted by 3: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Original message: HELLO

H → K
E → H
L → O
L → O
O → R

Encrypted message: KHOOR

Alice sends “KHOOR” across the network. Carol intercepts it but sees only “KHOOR”—she has no idea what it means.

Bob receives “KHOOR” and shifts in the opposite direction, back by 3:

Shifted alphabet:	D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
	↓ ↓ ↓ ↓
Original alphabet:	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Encrypted message: KHOOR

K → H
 H → E
 O → L
 O → L
 R → O

Decrypted message: HELLO

This is encryption. The message is scrambled using a rule (in this case, shift by 3 positions to the right on the letter’s position in the alphabet), and only someone who knows the rule can unscramble it.

The “rule” is called the **key**. In this case, the key is “3” (the shift amount). You can also think of the key as a password that both Alice and Bob know.

6.2 The Password Principle

Modern encryption works on the same principle, just with much more complex math. Instead of shifting letters, computers use mathematical functions that scramble bits in ways that are extremely difficult to reverse without the key.

Think of it like this:

$f(x)$ → means a math function

Like, for example: $f(x) = x + 2$

You've probably seen this in high school.

x can be any number. Some functions even have 2 inputs:

$f(x, y) = x + y + 2$

You choose the values of x and y (input, initial information), plug them into the function, and get an output (final information).

If you chose $x = 3$ and $y = 5$:

$f(3, 5) = 3 + 5 + 2 = 10$

Well, you can think of encryption like that too:

```
f(message, password) = scrambled_message
```

Encryption: Alice takes her message and a password, runs them through a function, and gets scrambled gibberish.

Decryption: Bob takes the scrambled message and the same password, runs them through the reverse function, and gets the original message back.

The reverse function is the function that does the “opposite.” If you add one, the reverse is to subtract one. Multiply by 3, then the reverse is divide by 3.

Example:

Message: "The password is banana123"

Password: "secret42"

Complex encryption function:

```
f("The password is banana123", "secret42") = "8x!mQ2$pL9@vN..."
```

Complex decryption function:

```
f_reverse("8x!mQ2$pL9@vN...", "secret42") = "The password is banana123"
```

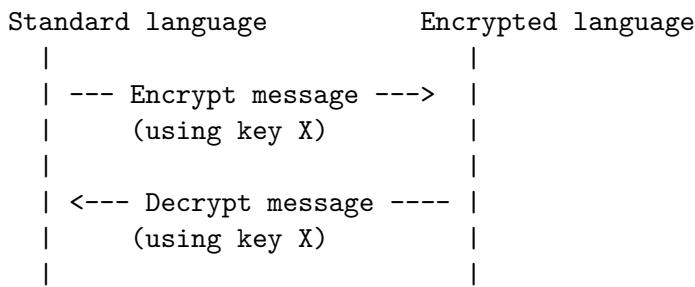
Carol intercepts "8x!mQ2\$pL9@vN..." but without knowing the password ("secret42"), she can't decrypt it. It's just random-looking noise to her.

This is symmetric encryption: Both Alice and Bob use the same secret key (password) to encrypt and decrypt.

But here's the catch: if Carol knew the function you use—the math operations involved—she could potentially break the encryption by just guessing passwords. It's easy to see why with the Caesar cipher, because there are only 25 possible shifts. So if Carol knows you're using a Caesar cipher, she can simply try shifting by 1. Still doesn't make sense? Okay, shift by 2. Still nothing? Try shifting by 3... BINGO! It makes sense. With only 25 possibilities, a computer can run through all of them very, very quickly.

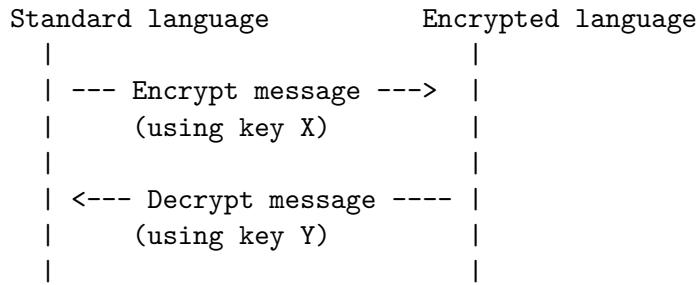
6.3 Why It's Called "Symmetric"

The name comes from the fact that the same key works in both directions:



Like a physical lock and key: - Alice locks the box with key X - Bob unlocks the box with the same key X - Both parties need the exact same key

This is different from **asymmetric encryption** (which we'll cover next), where Alice and Bob use different keys:



But that's for later.

6.4 Modern Symmetric Encryption

The Caesar cipher is easy to break—there are only 25 possible shifts, so Carol could try all of them in seconds.

Modern symmetric encryption uses much more sophisticated algorithms that do far more than just shifting characters around. A famous and widely used algorithm is:

AES (Advanced Encryption Standard): - Used everywhere: banking apps, HTTPS websites, encrypted files - Instead of shifting letters, it scrambles bits using complex mathematical operations

The point: With a strong password, symmetric encryption is virtually unbreakable by brute force. The Caesar cipher needs only 25 computations to crack, but with long enough passwords and a sequence of sufficiently complex operations, it would take billions of years for even the fastest computers to try all possible passwords (keys).

So symmetric encryption is actually used in networks, but in different complex and technical ways that you don't really need to understand here. For the nerdy or curious ones, research TLS (Transport Layer Security) and how it uses symmetric encryption for fast data transfer after establishing a secure connection. For normal people reading this, just continue reading.

6.5 The Fatal Flaw

So we've solved the problem, right? Alice and Bob can now communicate securely!

Not quite. There's a massive problem: **How do Alice and Bob agree on the password in the first place?**

Think about it:

Alice wants to send Bob an encrypted message.

But first, they need to agree on a password.

How does Alice tell Bob the password?

If she sends it over the Internet... Carol intercepts it!

Now Carol knows the password and can decrypt everything.

This is the chicken-and-egg problem of symmetric encryption:

- You have very nice math that allows for sending encrypted messages, but you need a shared password
- To share the password over long distances, you need... an encrypted channel
- But to have an encrypted channel, you need a shared password
- But to share the password... (infinite loop)

Note that at short distances Alice can just meet Bob in person and give him the password, but this isn't practical on the Internet where distances could span continents.

6.6 Long-Distance Relationships Did Not Work

Throughout history, people solved this by meeting in person:

Spies: Two agents meet in a dark alley, exchange code books face-to-face, then communicate securely via radio.

Military: Soldiers receive code books before deployment. If the enemy captures a code book, all communications are compromised.

Banking: You go to the bank in person, they give you a PIN, then you can use ATMs and online banking.

The pattern: **Pre-shared secrets.** Alice and Bob meet in a secure location beforehand and agree on a password. Then they can communicate remotely using that password.

But this doesn't scale to the Internet.

You can't fly to Amazon's headquarters to exchange a password before shopping online. You can't meet face-to-face with your bank before using their website. You can't physically visit Instagram's servers to set up encryption.

The Internet connects strangers who have never met and never will. How can they establish a shared secret over a public channel where attackers are listening?

For centuries, this seemed impossible.

Every encryption method required pre-shared secrets, and pre-sharing secrets requires a secure channel. But how do you create a secure channel without already having a shared secret?

It seemed like a logical impossibility—like asking someone to unlock a door when the key is inside the locked room.

This was the unsolved problem of cryptography until the 1970s.

6.7 Why This Matters for Bitcoin

You might be wondering: why are we talking about Caesar ciphers in a book about Bitcoin?

Because Bitcoin's security relies entirely on cryptography. Specifically:

Bitcoin addresses—your wallet—work with asymmetric cryptographic keys.

When you “own” Bitcoin, you don’t actually possess anything physical. You possess knowledge of a secret number (a private key, a password only you should know) that mathematically proves ownership. Without that key, the Bitcoin is mathematically inaccessible to you—and to everyone else. There you go: now you have a better understanding of what a Bitcoin wallet actually is.

And because it’s only information—a very big number used as an input in a math function—it can be stored anywhere: paper, hardware, titanium, your brain, etc. You choose the medium. A portable one? A durable one? It’s up to you and your needs. We’ll talk more about this later.

Bitcoin transactions are signed with private keys.

When you send Bitcoin, you’re creating a transaction and signing it with your private key. This signature proves: 1. You own the Bitcoin you’re spending. (Well, you own the mathematical secret that grants access to them.) 2. You authorized the transaction. (Because, allegedly, you did not share that number with anyone, so it must be you who is sending them.) 3. The transaction hasn’t been tampered with. (We will logically deduce this property later.)

But what does “sign” mean? In practice, it’s computing the mathematical function we saw earlier:

```
f(message, private_key) = encrypted_message.
```

Because this identifies you as the owner of the Bitcoin, we call it a signature—like signatures on classic paper bank checks or contracts. You can actually think of cryptocurrency signatures as classical banking checks. We’ll dive deeper later.

But how does this work over a public network where anyone can watch? How do you prove you own Bitcoin access without revealing your private key? How do thousands of strangers verify your signature (do the math computation) without knowing its input (your password)?

The answer resides in very special new properties you can achieve if you design the mathematical function cleverly enough—and that’s what we call asymmetric cryptography—the mathematical breakthrough that makes Bitcoin (and modern Internet security) possible.

But we had to understand symmetric encryption first, because it shows us the problem that asymmetric encryption solves. And because it’s relatively easy to explain with simple mental models—like the math functions we saw and the Caesar cipher.

So, summing up: now you know how to properly think about encryption—very complex mathematical functions that process information, mapping inputs to outputs. Helping computers instantly “invent” a new standard language only they know how to speak.

Depending on their properties, we classify them into symmetric or asymmetric encryption.

As we have seen, the properties of symmetric encryption are: - Both parties share the same secret key (password) - The same key is used to encrypt and decrypt messages

But this has a core complication: how do you share the secret key in the first place over long distances without someone intercepting it?

Key Insight: Symmetric encryption scrambles messages using a shared secret password. It’s unbreakable with modern algorithms like AES. But it has a fatal flaw: how do you share the password quickly and over long distances without someone intercepting it? Meeting in person

works, but doesn't scale to the Internet where billions of strangers need to communicate securely without ever meeting, and quickly.

Next, we'll explore the breakthrough that solved this: **asymmetric encryption**—a mathematical magic trick that lets you send secrets without sharing passwords, and prove identity without revealing secrets. This is one of the foundational blocks of Bitcoin and all modern digital security.

For now, we are starting to truly understand what a crypto-wallet actually is. It is just a password, a bunch of information—a very cleverly chosen one.

Chapter 7: Asymmetric Encryption - The Magic Trick and How They Create Crypto-Wallets

Two keys instead of one: a public key everyone can see and a private key only you know.

We've established the problem with symmetric encryption: Alice and Bob need to share a password before they can communicate securely, but sharing that password requires... a secure channel. Which requires a password. Infinite loop.

For centuries, this seemed impossible to solve.

Then, in the 1970s, mathematicians discovered something remarkable—a type of mathematical function with very special properties that seemed almost magical: **easy to compute in one direction, but extremely hard to reverse.**

This breakthrough is called **asymmetric encryption**, and it changed everything.

7.1 The One-Way Function

Let's start with the core concept: a **one-way function**.

Remember from Chapter 6, we talked about functions:

$$f(x) = x + 2$$

If $x = 3$, then $f(3) = 3 + 2 = 5$

This is an easy function. If you know the output (5) and the function used to compute it, you can easily figure out the input (3). Just subtract 2, one single operation, adding 2, which has a simple inverse operation, subtracting 2.

Challenge:

Bob used $f(x) = x + 2$ to cipher a message.

The result of ciphering the message was 5.

What was the original message?

We know then that:

$x + 2 = 5$

Which with simple math we can solve:

$x = 5 - 2$

$x = 3$

Answer:

Output = 5

Input/Message sent = 3

Reversing this function is very easy. For humans and for computers.

But what if we had a function that was easy to compute forward (input to output) yet extremely hard to reverse (output to input)? And we don't just mean "hard" like "takes a few seconds"—we mean "hard" like "**would take all the computers on Earth billions of years to reverse.**"

Does such a function exist?

Yes. Many of them, actually. They're the foundation of modern cryptography.

7.2 An Example For The Curious: Multiplication of Prime Numbers

Here's an intuitive example. If you feel confused, don't worry too much:

Easy direction: Multiply two prime numbers

Take two prime numbers: 89 and 97

Multiply them: $89 \times 97 = 8,633$

This is easy. A computer does this instantly.

Hard direction: Figure out the original prime numbers

Given: 8,633

Question: Which two prime numbers multiply to give 8,633?

This is much harder. You'd have to try dividing by every prime number until you find the original numbers.

Now imagine using **really, really big prime numbers**:

$p = 32,416,190,071$ (11-digit prime)

$q = 32,416,187,567$ (11-digit prime)

$p \times q = 1,050,807,929,418,200,854,057$

Good luck reversing that without knowing p and q!

Even computers take long to guess and check all possible combinations.

With very large primes (for example, 617-digit primes used in 2048-bit RSA keys commonly used in many traditional systems), **figuring out the original numbers becomes essentially**

impossible with current technology. Even the fastest supercomputers would take longer than the age of the universe to find such numbers.

Important note: Bitcoin doesn't actually use RSA or large prime factorization. Bitcoin uses **elliptic-curve cryptography (specifically secp256k1)**, which relies on 256-bit numbers (about 77 decimal digits)—a different type of one-way function based on elliptic curve mathematics rather than prime factorization. Both RSA and elliptic curve cryptography provide strong security through different mathematical problems that are easy in one direction but hard to reverse.

This asymmetry—easy to compute, hard to reverse—is the foundation of all asymmetric encryption, whether based on prime factorization (RSA) or elliptic curves (Bitcoin).

Why prime numbers? Because these numbers can only be divided by themselves and 1, leaving 0 as a remainder. This property is what makes the multiplication easy but the reverse operation hard. Don't worry too much about the mathematical details—if you're feeling confused about the prime number thing, take your time if you want, but this is not essential. You can just accept that there are some mathematical functions that are easy to compute one way but hard to reverse.

7.3 The Two Keys: Public and Private

Here's where it gets brilliant.

Step 1: Creating the keys (done locally on your computer)

Anyone can generate a public/private key pair using widely available algorithms (like RSA, ECC, etc). The algorithms are public knowledge in the same way anyone can describe and compute the function $f(x) = x + 2$ —you just need a computer to compute them quickly.

You run the computation on your computer:

```
generate_keys_computation() => (private_key, public_key)
```

The computation has that easy-only-one-way property we discussed: computing a public key from a private key (`f(Private key) -> Public key`) is the easy direction that anyone can compute, while the reverse (`f(Public key) -> Private key`) is the hard direction—mathematically incomputable with current technology.

So, you can compute a public key that is mathematically and uniquely associated with your private key, but no one can figure out your private key from your public key even if they know the algorithm (mathematical function) you used to calculate/compute it.

Step 2: Sharing the public key

Now you share your public key with everyone. Post it online, send it in emails, publish it in a database. It doesn't matter who sees it.

The setup:

Instead of one shared key (like symmetric encryption), asymmetric encryption uses **two different keys**:

1. **Public key** - You share this with everyone. Post it on the internet. Shout it from the rooftops. It doesn't matter who sees it. (The same thing you do with a crypto address)

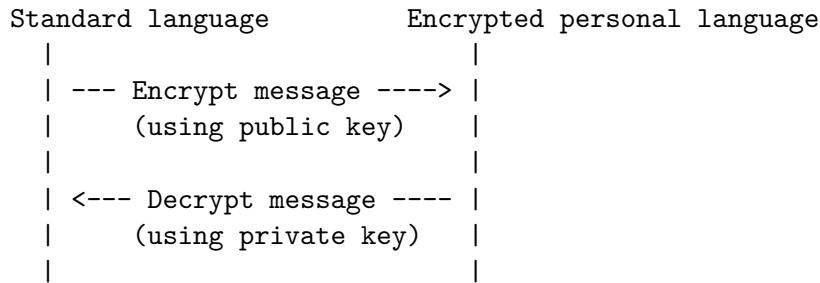
- 2. Private key** - You keep this secret. Never share it with anyone. Ever. (The same thing you do with a crypto wallet's private key, that 12 or 24 words sequence you might have heard of)

The magic: These two keys are mathematically related through a one-way function.

- You can **derive** the public key from the private key (easy direction)
- You **cannot** derive the private key from the public key (hard direction, which can create anonymity, security, and even identification properties if engineered for those purposes)

7.4 How It Works

Here's how communication works in the digital realm:



If anyone wants to talk to Bob and only to Bob, they encrypt their message with Bob's public key. Once encrypted, only Bob can decrypt it with his private key. Even if someone intercepts the message, they can't read it without Bob's private key.

And if Bob wants to communicate with anyone else, he can look up that person's public key and use it to encrypt his message. Only that person can decrypt it with their private key.

Just as a small reminder, you can think of encryption as translating from one language to another, like from English to a secret made up yet consistent language.

Imagine there is a public database with everyone's public keys. In practice it's not exactly like this, but imagine it—you can just look it up and send a private message to anyone without having to meet them or share a secret beforehand.

Example: Alice and Bob falling in love

Alice wants to say "I LOVE YOU, ALICE" to Bob, so she looks up Bob's public key and computes:

```
f("I LOVE YOU, ALICE", Bob's public key) = Encrypted message
```

Bob receives the encrypted message:

```
f_reverse(Encrypted message, Bob's private key) = "I LOVE YOU, ALICE"
```

Now Bob wants to say "I LOVE YOU TOO, BOB". He looks up Alice's public key and computes:

```
f("I LOVE YOU TOO, BOB", Alice's public key) = Encrypted message
```

Alice receives the encrypted message:

```
f_reverse(Encrypted message, Alice's private key) = "I LOVE YOU TOO, BOB"
```

If Carol intercepts the encrypted messages: - She sees the encrypted gibberish - She might even know Bob's public key - But she **cannot** decrypt the message without Bob's private key - And Bob's private key is mathematically impossible to derive from his public key

This solves the password-sharing problem.

Alice and Bob never had to meet. They never had to share a secret. Bob just published his public key, and Alice used it to send him a secure message—all over a public network where Carol is watching everything.

7.5 The Real Magic: RSA Encryption

The most famous asymmetric encryption algorithm is **RSA** (named after its inventors: Rivest, Shamir, and Adleman), which was invented in **1977** and published in **1978**.

For the curious ones who want to dig deeper into how RSA actually works mathematically, there are many resources online. But for our purposes, just know that it's a widely used algorithm that implements the one-way function property we've been discussing.

And for the curious ones who want to know how exactly you publish your public key in practice, it's usually done through **Public Key Infrastructure (PKI)** or **digital certificates** (like SSL/TLS certificates for websites). For Bitcoin specifically, your public key is related unequivocally to your Bitcoin address and revealed to the whole network when you make your first transaction.

7.6 Why This Matters for Bitcoin

Bitcoin doesn't encrypt transactions (they're public on the blockchain). What this means is that everyone can see that Bob wants to send X bitcoin to Alice, for example.

But Bitcoin uses asymmetric cryptography for something even more important—proving that Bob is the one who wants to send the transaction and proving that only Alice will receive them: **Proving ownership without revealing secrets**.

Remember from Chapter 6, your Bitcoin wallet is essentially a private key. But now let's understand the full picture:

The hierarchy:

```

Private Key (secret number you generate)
  | (one-way function)
  v
Public Key (derived from private key, can be shared)
  | (another one-way function)
  v
Bitcoin Address (where people send you Bitcoin)

```

Let's break this down:

- Your **Bitcoin wallet** is a **private key** (a very large secret number)
- Your **public key** is derived from your private key using a specific one-way function, like RSA.

- Your **Bitcoin address** (that long string of letters and numbers that represent your wallet and which people send money to) is calculated from your **public key** using another one-way function. This one is usually a function called a hash function.

Why this layered approach?

The wallet address is not the public key itself, but is derived from it. This creates cool properties like:

- It reveals less information about your public key, which, even though nothing bad can happen if someone figures it out, it is a cybersecurity best practice to just reveal as little information as needed. Details on this are unnecessary technical and beyond the scope of this book. As always, if you are curious, research.
- The resulting address is shorter and easier to share than the whole public key.

Example:

```
Public key: 04b0bd634234abdc04b0bd634234abdc04b0bd634234abdc... and more
Bitcoin address: 1A1zP1eP5QGefi2DMPT (smaller and also unique)
```

[these numbers were made up, it is just a visual representation]

If this represents a number, why do I see letters? Because this is written in hexadecimal format.

We humans use the decimal format, based on 10 digits (0-9). Computers often use binary format (based on 2 digits: 0 and 1). Hexadecimal format is based on 16 digits. But we only have 10 digit symbols (0-9), so we start using letters when we run out:

UNIQUE SYMBOLS

Decimal: 0 1 2 3 4 5 6 7 8 9

Binary: 0 1

Hexadecimal: 0 1 2 3 4 5 6 7 8 9 A B C D E F

If you don't really understand this, don't worry. Just think that those letters, somehow, also represent numbers.

When you spend Bitcoin, you create a **digital signature** using your private key to signal that you authorize the transaction. The only “digital” part of it is that it’s done on a computer, but a more accurate term would be a “mathematical signature.”

The signature proves: 1. You own the Bitcoin (you possess the private key) 2. You authorized the transaction, only if point number 1 is true and you are the only one who has the private key, this is where personal responsibility comes in. 3. The transaction hasn’t been tampered with. This does not depend on you, this is a mathematical property.

The magic: - Anyone can verify your signature using your public key - But only you can create the signature (requires your private key) - And no one can figure out your private key from your public key

The World

Bob's computer

```
|           |
|           |
|           | sign("I send 5 BTC to Alice")
|           | [using private key]
|           |
```

```

|      (      Message sent      )      |
| <---( "I send 5 BTC to Alice", )--- |
|      (      output from signing   )      |
|
|
|
```

As per the image above, now The World has 2 things: 1. The message: “I send 5 BTC to Alice” 2. The output from signing (the signature) that message with Bob’s private key

If they want to make sure that only Bob could have created that message, they can use Bob’s public key to verify the signature.

If Bob created the message, then using the “reverse math” to the sign should give the message. This means:

```
f_reverse(output from signing, Bob's public key) = "I send 5 BTC to Alice"
```

So the world can just do the reverse math, and if the message matches, it can only mean one thing: this message was produced by someone who has Bob’s private key, which is only Bob.

The process has been slightly simplified, the verification process of the signature is a bit more complex but this mental model works for the intents and purposes of this book. In reality the function looks more like this:

```
f_verify(signature, public key, original message) = True/False
```

This is **asymmetric cryptography in reverse**: instead of encrypting messages, you’re signing messages and sending the signature with them. Instead of “only Bob can decrypt,” it’s “only Alice could have signed this, and everyone can verify it, because the math is public—everyone knows RSA for example.”

We’ll explore digital signatures in depth in the next chapter. Understanding the difference between crypto-signatures and crypto-transactions is crucial for operating with cryptocurrencies safely.

7.7 Cool Properties

Let’s summarize what we’ve learned:

- Anyone can generate a public/private key pair using widely available algorithms (like RSA, ECC, etc)
- The algorithms are public knowledge; you just need a computer/phone to compute them
- You can share your public key with anyone, and your crypto wallet address too. It is better if you only share the wallet address, but sharing the public key is also safe—no one will steal money from you just with that
- The world can use your public key to send you encrypted messages or verify your digital signatures
- But only you can decrypt those messages or create signatures, because only you have the private key

7.8 What We've Solved

Let's review what asymmetric encryption enables us: the generation of unique and globally valid passwords, secure communication over public channels, and proving identity without revealing secrets.

These properties enabled stuff like: - Secure online banking - E-commerce - Distance agnostic encrypted messaging - **And cryptocurrency wallets**

All because we have figured out a way to create homemade IDs. A large unique number impossible to guess associated with another one you can share, and which anyone can verify that only you created that shared number, associating its creator with it. Or in other words, identifying its anonymous creator with it. All assured thanks to math with that uses the easy-to-compute-only-one-way property.

7.9 Summary

If even with this level of simplification this is still hard to grasp, do not worry, it is normal. I'm trying to explain things so you feel safer because you understand the dynamics to some extent. If this still feels too complex, just remember one simple thing:

- IN GENERAL, NEVER SHARE YOUR PRIVATE KEY WITH ANYONE.
 - NOT YOUR PRIVATE KEY, NOT YOUR COINS.
 - IF SHARING, YOU ARE GRANTING ACCESS TO THEM. MAKE SURE IT IS ONLY TO PEOPLE YOU TRUST 101%.
-

Key Insight: Asymmetric encryption uses two keys instead of one. A public key (shared with everyone) and a private key (kept secret). Messages encrypted with the public key can only be decrypted with the private key. The keys are mathematically related through one-way functions—easy to compute forward, essentially impossible to reverse. This solves the key-sharing problem: strangers can communicate securely without ever meeting or pre-sharing secrets.

Now that you understand asymmetric cryptography—the mathematical foundation of cryptocurrency wallets and secure internet communication—there's an important question we need to address: **“Won’t quantum computers break all of this?”** This is one of the most common concerns about cryptocurrencies. In the next chapter, we'll briefly explore what quantum computing actually is, what it can and can't break, and why this isn't an apocalyptic scenario for crypto as some imagine.

8

Chapter 8: Quantum Computing - The Future Threat?

Will quantum computers break everything we just learned?

Now that you understand asymmetric cryptography—the mathematical foundation of cryptocurrency wallets and secure internet communication—there’s an important question we need to address:

“Won’t quantum computers break all of this?”

This is one of the most common concerns about cryptocurrencies. Let’s tackle it head-on.

8.1 The Concern

Asymmetric cryptography relies on one-way functions—mathematical operations that are easy to compute forward but extremely hard to reverse. For example, multiplying two large prime numbers is easy, but figuring out which two primes were multiplied is extremely hard (it would take classical computers billions of years). Here’s the concern: **quantum computers might be able to reverse these “one-way” functions much faster.**

8.2 What is Quantum Computing?

Remember from Chapter 1: a classical bit is either 0 or 1—a switch that’s either off or on.

A **quantum bit** (or **qubit**) is different. Due to bizarre physical phenomena at the atomic level and extreme low temperatures, a qubit can exist as **0 and 1 at the same time**. This is called **superposition**.

Classical bit:

[0] or [1] (one state at a time)

Quantum bit (qubit):

[0 AND 1 simultaneously] (superposition of states)

When you have multiple qubits in superposition, they can represent many possible combinations at once, allowing quantum computers to explore many solutions simultaneously and making certain types of calculations exponentially faster.

Important: Quantum computing is not magic. It doesn't make everything faster—it's very good at specific types of problems (like factoring large numbers) but not universally better at all computing tasks.

8.3 What Breaks and What Doesn't

What quantum computers could break: - RSA (relies on factoring large numbers) - Elliptic Curve Cryptography / ECDSA (used by Bitcoin and Ethereum) - Timeline: Experts typically estimate that quantum computers capable of breaking today's public-key cryptography are **10–20 years away**, with some putting the risk around the **2030s**, though uncertainty and disagreement remain about exact timelines.

What stays secure: - Symmetric encryption like AES (just use larger keys) - Hash functions (relatively resistant) - Post-quantum cryptography algorithms (already exist and were standardized by NIST in **August 2024** with the release of FIPS 203, 204, and 205, covering algorithms like Kyber, Dilithium, and SPHINCS+)

8.4 Why Cryptocurrencies Can Adapt

Cryptocurrencies are software. Software can be updated.

Bitcoin uses ECDSA for signatures, but there's nothing preventing a switch to quantum-resistant algorithms. The core protocol—blocks, transactions, consensus—remains the same; only the signature algorithm changes.

Example migration:

Current: Private Key → (ECDSA) → Public Key → Bitcoin Address

Future: Private Key → (Post-quantum algorithm) → Public Key → New Bitcoin Address

Users would generate new quantum-resistant wallets and transfer their funds. The blockchain continues, just with new signature algorithms.

8.5 Everyone Has This Problem

If quantum computers break cryptocurrency encryption, they break EVERYTHING:

- Online banking → broken
- Military communications → broken
- Government secrets → broken
- Every HTTPS website → broken

Cryptocurrencies are not uniquely vulnerable. They face the same quantum threat as everyone else—and they're actually MORE adaptable because crypto protocols can update via consensus, while traditional systems are notoriously slow to change.

The entire world has incentive to develop quantum-resistant cryptography BEFORE powerful quantum computers exist. And that work is already well underway.

8.6 The Treasure Hunt: Satoshi's Bitcoin

Here's a fascinating consequence:

Satoshi Nakamoto, the pseudonymous creator of Bitcoin, mined between **750,000 and 1.1 million Bitcoin** in the early days (most analyses estimate around **1 million BTC**)—today worth tens of billions of dollars.

When you spend Bitcoin, you reveal your public key on the blockchain. Satoshi made some transactions, revealing some public keys.

If someone builds a quantum computer powerful enough to break ECDSA, they could derive private keys from those revealed public keys and steal the Bitcoin. This creates two possible outcomes:

1. Whoever breaks ECDSA first finds the “treasure” (billions of dollars in early Bitcoin)
2. The network agrees to move all those funds to a new quantum-resistant address controlled by no one

This isn’t just theoretical—it creates real incentive to solve quantum threats before they arrive. With billions of dollars at stake, people take the threat seriously and will migrate in time.

Key Insight: Quantum computers pose a future threat to some cryptographic algorithms, including those used by cryptocurrencies. However, quantum-resistant algorithms already exist, and migration is possible. The entire internet faces this same challenge, creating strong incentives for timely solutions. This is a manageable transition, not a catastrophe.

Now that you understand the cryptographic tools—how to create digital IDs, prove ownership, and sign transactions—let’s explore what comes next: **Can we make bits mean “money”?** In the next chapter, we’ll see how bits can have the properties of money, why we need a distributed database to store them, and the fundamental challenge that makes cryptocurrencies possible: the consensus problem.

9

Chapter 9: Money as Synchronized Bits

If we can assign meaning to bits, why not assign them the meaning of money?

You now understand that cryptography lets you create digital IDs from your home and prove your intent with signatures. You know that information is just bits, and humans assign meaning to those bits.

Here's the natural question: **Can we make bits mean “money”?**

The answer is yes. But first, let's understand what “money” actually needs to be.

9.1 What Makes Something Money?

Throughout history, humans have used all sorts of things as money: salt (so valuable the word “salary” comes from it), shells (used across continents for millennia), gold and silver (heavy, but universally valued), paper bills (convenient, but only valuable because we agree, and taxes), and digital bank account numbers (just entries in a database).

What do all these have in common? They share certain properties that make them useful as money.

9.1.1 Property 1: Scarcity (You Can't Create It Easily)

If I have 10 units of money, I can't suddenly have 11 without earning it, finding it with effort, or someone giving it to me.

Examples: - Gold: Hard to mine, can't just create more. - Dollar bills: Government controls printing (can't photocopy and use it). - Salt (historically): Required effort to extract from sea water.

Counter-example: - Leaves: Too abundant, everyone would be “rich”. - If anyone could create money freely, it would be worthless.

There are more philosophical and sociological reasons to justify that for something to be valuable it needs to be scarce. This book is not that much about social coordination but rather the explanation of how new technologies can be used for it. Therefore here I leave a simple intuition and argument explaining why value must have some degree of scarcity:

For something to be valuable it needs to be difficult to create and scarce. Money is the abstraction of value so we can coordinate trade. If someone could just gather 100 leaves and say “I now have power over what you have to do for me,” the system breaks down. We would just gather leaves non-stop. In order to reward work that actually creates value, we cannot just give value to anything that exists in accessible abundance.

For bits to be money: We need rules preventing uncontrolled creation. The system must enforce hard-to-control scarcity.

9.1.2 Property 2: Verifiability (You Can Prove What You Have)

If I claim to have 10 units, I need to be able to prove it to you, and you need to be able to verify it's real.

Examples: - Gold: You can weigh it, test its purity. - Dollar bills: Security features (watermarks, special paper). - Bank account: You can check your balance, show a statement.

Counter-example: - If I just say “I have 10 units of value,” but can't prove it, you won't accept it.

For bits to be money: We need a way to check balances. Some shared record everyone can verify, look up, and preferably, instantly.

9.1.3 Property 3: No Double-Spending (Can't Spend the Same Money Twice)

If I have 10 units and give them to you, I no longer have them. I can't also give those same 10 units to someone else.

Examples: - Physical cash: Hand you a \$10 bill, and I don't have it anymore. - Gold: Give you the gold coin, and it's physically with you now. - Bank transfer: Bank deducts from my account, adds to yours.

Counter-example: - If I could spend the same \$10 with you AND with Bob, money breaks. - This is the tricky part with digital files (can copy infinitely).

For bits to be money: When I transfer them to you, those specific bits must no longer be “mine.” The system must track ownership clearly.

9.1.4 Property 4: Transferability (You Can Send It)

Money is useless if you can't give it to others in exchange for goods, services, or other money.

Examples: - Cash: Hand it over, immediate transfer. - Bank transfer: Send it electronically, through the internet we now understand. - Gold: Can transport it (though heavy).

For bits to be money: We need a mechanism to change ownership. A way to say “these bits were mine, now they're yours.”

9.1.5 Property 5: Ownership (You Actually Control It)

You need to truly possess your money. Not just “have permission to use it” from someone else.

Examples: - Cash in your pocket: YOU control it physically. - Gold you're holding: YOU possess it. - Bank account: Well... the bank actually controls it (you just have permission).

The bank account problem: You don't hold the money; the bank does. You trust them to give it back when you ask. They can freeze your account, deny access, or even go bankrupt. Your "money" is just their promise to you.

For bits to be money: Ideally, YOU control them directly. Like cash in your pocket, but digital.

9.1.6 Property 6: Fungibility (Each Unit is Equivalent)

One unit of money should be the same as any other unit. They're interchangeable.

Examples: - Dollar bills: Any \$10 bill = any other \$10 bill. - Gold: An ounce of pure gold = any other ounce of pure gold (by weight and purity).

Counter-example: - Unique collectibles: Each is different, not interchangeable.

For bits to be money: 1 unit of bit = 1 unit of bit. Doesn't matter which specific bits you have.

9.1.7 Property 7: Divisibility (Can Break Into Smaller Parts)

Sometimes you need to pay less than 1 full unit. Money should be divisible.

Examples: - Dollar: Can be divided into cents (\$0.01). - Gold: Can be cut or melted into smaller amounts. - Bitcoin: Divisible to 8 decimal places (0.00000001 BTC, called a "satoshi").

For bits to be money: Should support fractions. Not just whole numbers.

9.1.8 Property 8: Durability (Doesn't Disappear or Decay)

Money should last. If it degrades quickly, it's not useful for storing value.

Examples: - Gold: Doesn't rust or decay (lasts forever). - Coins: Metal endures for decades. - Paper bills: Get worn out, but last a few years.

Counter-example: - Food: Rots, can't be used as money.

For bits to be money: Digital information doesn't decay (can copy perfectly), but you need to make sure it doesn't get lost. If you lose access (lose your password/keys), it's gone.

9.2 The Realization: Bits Can Have These Properties

Look at what we've learned so far:

From Chapter 2 (Logos): Humans assign meaning to bit patterns. We've assigned meaning to bits as letters (ASCII), bits as images (JPEG), and bits as music (MP3). Why not bits as money?

From Chapter 7 (Asymmetric Cryptography): We can create digital IDs (public/private keys) and prove ownership with signatures. This solves: - **Ownership:** Your private key = your control (like cash in your pocket, digitally). - **Transferability:** Sign a message saying "I send X to Bob" and it proves intent.

What's missing? The coordination part.

9.3 The Database Problem

Think about it simply: if bits represent money, where do you store them?

Bits are information, and information needs storage. Information is stored in databases, which are ultimately those physical devices called transistors that have 2 possible states, each one representing a bit, like a door. But as we said, doors are very slow and clunky transistors. It is better to just use computers.

So we need a database—just a computer storing information in bits—that tracks who has how much money (balances) and who sent money to whom (transactions).

Simple example:

Database:

- Alice: 50 coins
- Bob: 30 coins
- Carol: 20 coins

Transaction:

- Alice sends 10 coins to Bob

New data (state) of the database:

- Alice: 40 coins
- Bob: 40 coins
- Carol: 20 coins

Easy, right?

9.4 The Traditional Solution: One Database, One Controller

This is how banks work:

The bank's database: - Stores everyone's balance. - The bank controls the database. - When you “send money,” you ask the bank to update the database. - Bank checks: Does Alice have 10 coins? Yes? Okay, update: - Subtract 10 from Alice. - Add 10 to Bob. - Done!

This works. One central point to look at, to interact with, centralized. But notice the problem:

You don't control the money. The bank does.

The bank can freeze your account, deny your transaction, or go bankrupt (your money disappears). You must trust the bank to maintain accurate records and give you access when you need it.

Remember **Property 5 (Ownership)**? You don't truly own your money. You have a balance in someone else's database.

9.5 The Insight: Distribute the Database

Here's the idea: **What if we gave EVERYONE a copy of the database?**

Instead of one bank holding the records, Alice has a copy of the database, Bob has a copy, Carol has a copy, and thousands of other people have copies too.

Now: - No single person controls it (de-centralized). - Alice can verify Bob's balance herself (just check her copy). - No one can secretly change the records (everyone would notice).

This solves some problems: - **Verifiability:** Anyone can check anyone's balance. - **Ownership:** No single entity controls access, just you and your private keys. - **Transparency:** All transactions are visible.

But this still has problems... For example, it doesn't solve the double spending problem by itself. It does not have property 3 yet.

9.6 The Consensus Problem

Here's where it gets tricky.

If everyone has a copy of the database, how do we ensure all copies stay synchronized as the data changes over time? Since the bits in our database are being interpreted as money, we can rephrase this: how do we ensure all copies stay synchronized as money is being transferred between people?

Problem 1: Conflicting updates

Alice sends 10 coins to Bob (signed transaction)
 Bob receives it, updates his database: Alice: 10, Bob: 40

At the same time, Carol receives a different transaction:
 Alice sends 10 coins to Carol (also signed by Alice)

Bob's database: Alice: 0, Bob: 40, Carol: 20
 Carol's database: Alice: 0, Bob: 30, Carol: 30

Which is correct? They both have valid signatures, and in both Alice had 10 coins to spend when she sent them.

How do we prevent this? Who decides which transaction is valid? Both cannot be valid, because Alice would be double spending her 10 coins.

You can say, send it first to Bob and then to Carol, but what if the messages arrive at the same time? Remember we are on the internet—it is like the digital wild west, with no central authority to decide who gets to write first or receive the message first.

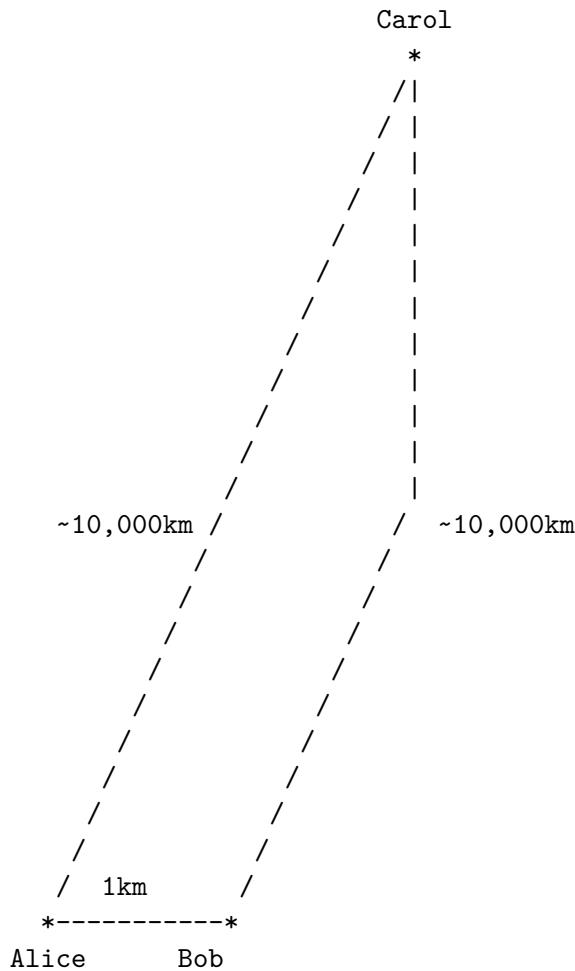
Well then let's create a consensus of always sending the messages in that order and we will not have problems? It might work for 3 people who trust each other, but for thousands of people who should not need to trust each other, how do we ensure everyone agrees on the same order of transactions?

1000 computers all have the database
 Alice signs: "I send 10 to Bob"
 Who gets to add this to the database?
 Do all 1000 computers add it at the exact same time?
 What if someone adds fake transactions? What if Alice tries to confuse the network holding the database?

Problem 2: Latency (Distance Delays)

Even if everyone is honest, physical distance creates synchronization problems.

Imagine Alice lives 1km away from Bob, and Carol lives 10,000km away from both. Carol will pay Alice 5 coins, and Alice wants to buy something from Bob for 10 coins. Alice only has 8 coins, so she cannot buy it yet. But then Carol sends her 5 coins, giving Alice enough to buy from Bob.



- If everyone has a copy, they all need to agree.
 - How do we synchronize thousands of databases?
2. **How do you make sure no one transfers or creates money without permission?**
 - We have signatures (good), but how do you enforce network participants check them?
 - What if someone tries to spend money they don't have?
 - What if someone tries to spend the same money twice?
 3. **How do you reach consensus on how and with what limits to write in this shared database?**
 - Who gets to add new transactions?
 - How often do we carry updates?
 - What rules must everyone follow?
 - How do we enforce those rules without a central authority?

This is the consensus problem.

And solving this problem—enabling thousands of strangers to maintain an identical database without trusting any single authority—is what makes Bitcoin and other cryptocurrencies possible. It is, as you can see, a very hard problem.

9.8 Historically Speaking

For thousands of years, humans used physical objects as money. Salt was valuable because it preserved food (essential for survival). Shells were rare, beautiful, and hard to counterfeit. Gold was scarce, didn't decay, had universally agreed value, and was divisible. Paper was convenient but required trusting the issuer (government and bank). Each era found something that worked for the technology and trust models of the time.

Physical money had built-in scarcity (hard to create) and ownership (possession = ownership).

Digital money with banks worked by trusting a central authority to maintain the accounting and to maintain money's value stable.

But consensual synchronized bits—distributed across thousands of computers, with no central authority, where YOU control your money with cryptographic keys—this is new.

It's only possible because Satoshi (whoever he, she, or they are) introduced the first **practical, decentralized consensus mechanism for digital money**—often called *Nakamoto consensus*. This solved the double-spend problem in an open network by combining proof-of-work, longest-chain selection, and economic incentives. It's not *the* final solution to all consensus research (that's a broader field with many variants), but it was the breakthrough that made Bitcoin work.

Key Insight: Money is just information we agree has value. Bits can represent money if they have the right properties: scarcity, verifiability, no double-spending, transferability, ownership, fungibility, divisibility, and durability. We can store this information in a database and distribute copies to thousands of computers. But to make it work, we need to solve a hard problem: How do thousands of strangers agree on the same database state (data) without trusting any central authority? That's the consensus problem—and that's what we'll explore next.

By the way, when I say state it means the same as the current data in a database, the current information the database holds.

Next, we'll see how **consensus mechanisms** solve these problems. We'll understand Proof-of-Work, mining, and why the blockchain structure makes history tamper-evident. The pieces are coming together.

10

Chapter 10: Proof-of-Work - Earning the Right to Write

What if we borrowed hierarchy temporarily, but earned leadership through effort?

We've established the problem: distributing a database across thousands of computers creates synchronization chaos. Conflicting updates, latency delays, and malicious actors can all threaten to break the system.

Here's a natural thought: **What if we temporarily gave someone the power to decide?**

10.1 Borrowing Hierarchy (Just for a Moment)

Think about it: many of the problems we identified in Chapter 9 get easier if one person has authority:

Problem 1 (Conflicting updates): If Alice sends 10 coins to Bob AND to Carol at the same time, who decides which transaction is valid? Easy—the person in charge decides. They pick one, reject the other. Done.

Problem 2 (Latency delays): If databases are temporarily out of sync because information takes time to travel, who decides what the “correct” state is? The person in charge waits a second or two and declares: “This is the official version.” Everyone updates to match.

Problem 3 (Security): If Dave tries to flood the network with fake transactions or send conflicting data to different people, who stops him? The person in charge validates transactions and only includes the legitimate ones.

Centralization makes coordination way simpler—this is why banks work so efficiently. One authority, one database, one source of truth.

But we don't want permanent centralization. That brings us back to the bank problem—lack of ownership, frozen accounts, control by a single entity.

The insight: What if we give someone temporary authority to write to the database, but: - Only for a short period (not forever like banks). - They must follow rules (like “no double-spending”). - We rotate who gets this power (not the same person every time).

This is the elegant compromise: Borrow hierarchy temporarily, but distribute it over time.

10.2 Who Gets the Right to Write?

Okay, so we'll give someone temporary power to add transactions, to write new information to the database. But who?

We can't just vote. IDs here are created from home—Dave could run 500 fake identities and outvote everyone. One computer, one vote doesn't work when anyone can pretend to be 500 computers. Furthermore, we do not know each other, and we do not need to. How are you going to vote for anyone worthy of trust if you do not know them?

We can't just pick randomly. Same problem—if Dave runs 500 computers in a 700-computer network, he has a 500/700 chance of being chosen compared to 200 honest participants. He wins most of the time. What if he is evil? The evil wins most of the time? That is, by definition, not good.

We need a different selection mechanism.

10.3 How Humans Choose Leaders

Throughout history, humans have chosen leaders based on **feats**: - The brave hero who defended the village (courage demonstrated through action). - The super-intelligent doctor who invented a cure (expertise demonstrated through achievement). - The skilled craftsman who built the best tools (competence demonstrated through work).

Feats require: 1. **Skill** - You need to know how to do something. 2. **Effort** - You need to actually do the work. 3. **Proof** - Others can verify you accomplished it.

You can't fake a feat easily. You can't claim to be a hero without fighting. You can't claim to have invented a cure without showing the working medicine. The feat itself is proof.

What if we used this same natural principle for choosing who gets to write to the database?

Require a feat—something that requires effort, can be verified, but can't be easily faked.

10.4 The First Digital Feat: Proof-of-Work

In the physical world, feats involve courage, intelligence, or craftsmanship. In the digital world, what's the equivalent?

Computation, run by electricity, energy.

Computers do work by computing—running calculations, processing data, solving math problems. Some computations are easy (adding 2+2), while others are very hard (finding a specific pattern in billions of random numbers).

Here's the idea: To earn the right to write the next batch of transactions to the database, you must solve a difficult computational puzzle.

Not a puzzle that requires cleverness (that would favor smart people unfairly, and it is very hard to design). A puzzle that requires **brute force** would do better—trying billions of guesses until you find the right answer. This consumes real electricity, electricity that you could have used for

something else, like warming your home or running your business. So, if you really want to use that electricity to earn the right to write to the database, you have to make a sacrifice. Will people agree that this sacrifice is worth it? Turns out they did, and still do.

But if they did not, even if you were following the rules and solving the agreed puzzle, all your effort and electricity would have been wasted. You, the worker and potential future leader, and the others have a mutual dependency on each other. I do the work, and I trust that you guys will value it. And vice-versa, you have proven to do the work, and I reward you by letting you write to the database and me using it. But remember, tomorrow it might be someone else, an evil one even, so you better follow the rules and keep working.

Not only do the database users depend on the miners, but the miners depend on the users finding the database useful too. Brilliant game theory dynamics at play here.

Miners are just a special name put on the computers on the network who are trying to solve these puzzles to earn the right of writing the next transactions to the database.

Bitcoin's algorithm that showcases their interpretation of effort is called: Proof-of-Work.

10.5 The Puzzle

Let me explain the puzzle Bitcoin uses (simplified):

The goal: Find a number that, when combined with the transaction data and put through a hash function, produces an output starting with a certain number of zeros.

What's a hash function? Remember one-way functions from Chapter 7? A hash function is similar—it takes any input and produces a fixed-size output that looks random. Change the input even slightly, and the output changes completely.

Example (simplified):

```
hash("Hello") = d3a1f2
hash("Hello!") = 9f4e7b (completely different, and... unique too! if long enough)
```

The hash function is a one-way function that is also deterministic (same input always gives same output), but unpredictable (you can't predict the output without computing it).

Because hash functions are unpredictable and produce outputs of sufficient length, you can use them to generate IDs—the likelihood of two different inputs producing the same output is astronomically low. Designing a good hash function is not trivial, but for our purposes, we will assume Bitcoin's hash function (SHA-256) is good enough.

The puzzle:

Input: [Transactions + Previous Block + Random Number You Chose]
Output: Hash that starts with, say, 5 zeros (00000...)

Try:

```
hash([Transactions + Previous Block + 0]) = 3f2a8c... (nope, doesn't start with 00000)
hash([Transactions + Previous Block + 1]) = 7d1b5e... (nope)
hash([Transactions + Previous Block + 2]) = 5c8f2a... (nope)
```

...
`hash([Transactions + Previous Block + 8,347,291]) = 00000f8a2c... (YES! Found it!)`

The only algorithm to solve this is... brute force. Your only option is to try billions or trillions of random numbers until you find one that produces a hash starting with the required number of zeros.

This is the feat. It requires: - **Effort:** Trying billions of guesses requires massive computational work (electricity, time, hardware). - **Proof:** Once you find the answer, anyone can verify it instantly (just run the hash function once with your number). - **Can't be faked:** You can't pretend you did the work without actually doing it, because the hash function is one-way. You cannot just have a value with 5 leading 0s and try to guess the input that created it.

10.6 Why This Works

Let's see how Proof-of-Work solves our problems:

1. **Who gets to write?** Whoever solves the puzzle first. They earned the right through effort.
2. **Can Dave cheat with 500 fake identities?** No. Running 500 computers doesn't matter—what matters is computational power. If Dave has 30% of total computing power, he wins approximately 30% of puzzles, while honest participants with 70% of computing power win approximately 70% of the time. It's proportional to effort, not number of identities.
3. **How do we know they followed the rules?** When someone publishes their bunch of transactions (called a “block”), everyone else verifies: - Did they solve the puzzle? (Check the hash validity.) - Are all transactions valid? (Check signatures, check balances.) - No double-spending? (Check against the database.)

If anything is invalid, everyone rejects the block, all the new transactions, and does not update (synchronize) their database. The cheater wasted all that computational effort for nothing.

4. **How often does this happen?** Bitcoin adjusts a **target hash value** so a new block is found approximately every 10 minutes. On screen, this often looks like requiring hashes with **more leading zeros**. If more people join and computing power increases, the target gets lower (harder to meet). If people leave and computing power decreases, the target gets higher (easier to meet).

The reason you can “fix” the time it takes to mine (create) a block is because you can do some statistics to compute the expected time to find a solution based on the total computational power of the network. By adjusting the difficulty of the puzzle every 2016 blocks (approximately two weeks), Bitcoin ensures that, on average, a new block is found every 10 minutes. If you are interested in the math behind it, I encourage you to research online and dive deep on your own.

For now, let's demystify some terminology: - **Mining:** The process of trying to solve the puzzle (running computations). A proof of effort to earn the right to write the next bunch of data on the distributed database you all share in the network. - **Miner:** A participant who runs computers to mine (solve puzzles). - **Block:** A batch of transactions plus the solution to the puzzle. Which, generally speaking, are a set of instructions on what to write next in the database. - **Node:** A participant who maintains a copy of the database and verifies blocks (not necessarily mining). They can accept or reject blocks based on the rules that have been agreed upon. If the miner shares invalid blocks, nodes will reject them, and everyone will just have the same database state. Valid?

Everyone updates the database adding the transactions in the order set by the temporary leader and thus all end up with the same state on the database.

10.7 The Incentive: Mining Rewards

Wait—why would anyone spend electricity and money on expensive computers to solve these “useless” puzzles?

Because there's a reward.

Whoever solves the puzzle gets to: 1. **Create new coins** - Currently 3.125 Bitcoin per block (this halves every 4 years, eventually reaching zero around the year 2140, if the consensus does not change). 2. **Collect transaction fees** - Users can attach small fees to their transactions. The miner who includes them in a block collects all those fees.

This is called mining. Like gold miners expending effort to extract gold from the ground, Bitcoin miners expend computational effort to earn Bitcoin. And, as Bitcoin shares a good bunch of properties gold has, I guess that is why Satoshi Nakamoto called it “mining”. But this is just the author’s speculation.

The incentive aligns perfectly: - Miners want rewards, so they follow rules (invalid blocks get rejected, wasting their effort). - Miners compete, so they invest in more computing power (securing the network). - As Bitcoin’s value increases, more miners join, which means more security.

Game theory in action: It’s more profitable to play honestly and keep the network running than to cheat.

10.8 The Miracle of Bitcoin, the Wet Dreams of Nerds, Maybe

But wait... they get paid... in what? In bits? Indeed. In bitcoins! But no one cared about these bits initially. Indeed, here is where “the miracle” happened.

A group of people, likely computer science and anarchist “nerds” (said with love by the author, who is a nerd himself), decided to just spend their electricity and computing power on this “crazy experiment” of a decentralized digital currency. They believed in the idea, or they were just nerds having fun, and they were willing to invest resources into it.

Some people speculate it was the CIA or any other United States government agency trying to create programmable money to control the population. But as said, Satoshi Nakamoto disappeared from public view—his **last public forum post was in December 2010**, and his **last known private email was in April 2011**—and no concrete evidence has ever been found to support any theory about his/her/their identity or affiliation.

The author leans towards the nerds playing around—maybe CIA nerds on their weekends? Who knows. The thing is that the party went out of control, and slowly but surely, more people agreed on this digital coordination and consensus and started actually trading these bits with fiat money. And... boom! A new asset class was born.

It is quite poetic that the thing which is valuable precisely because people can use it without even knowing each other was created by someone who no one knows—in practice then, by no one.

Therefore, poetically, we can dare to say that it was created by no one, and by all of us, at the same time.

But wait! As per all we've explained, we can still cheat the network. Notice that we did not say the word "blockchain" at all...

10.9 Temporary Leadership

Notice what we've achieved:

- **Every 10 minutes**, someone earns the right to write the next batch of transactions to the database.
- **They have temporary authority** - But only for that one block. Then the race starts again.
- **Leadership rotates** - Whoever solves the next puzzle becomes the next temporary leader.
- **No permanent power** - No single entity controls the database forever... or does it? If someone has 51% of all the computational power, they will just be more likely to write more blocks, but they still have to follow the rules or everyone else will reject their blocks.

We've borrowed hierarchy temporarily, distributed it over time, and used effort as the selection mechanism.

This is radically different from banks: - Banks: One entity has permanent control. - Bitcoin: Temporary control, earned through computational work, rotating among thousands of participants.

10.10 What We've Solved (Have We?) and What We Haven't

What Proof-of-Work solves: - Who can write: Whoever solves the puzzle first. - Sybil attacks: You need computational power, not fake identities. - Ordering transactions: The winner decides the order in their block. - Incentives: Miners are rewarded for honest work. Sybil attacks do not make sense for agents inside the network anymore.

What we still need to address: - How are we explaining blockchain without the very blockchain structure itself? Why do we need it? (Next chapter.) - What happens if two miners solve the puzzle at the same time? What happens if this happens often? - How does this make history tamper-evident? - What if a miner controls more than 50% of the computational power? Do they gain too many chances of having the right to write on the database?

10.11 The Bigger Picture

Proof-of-Work is Bitcoin's way of selecting temporary leaders based on effort. It's not the only consensus mechanism, or consensus algorithm if you prefer. Ethereum, another "crypto," now uses Proof-of-Stake, which we'll explore later. But Bitcoin was the first to successfully solve the distributed synchronized database network problem without trusted middlemen, using consensus instead.

The breakthrough: Using energy (computational work) as a relatable and scarce resource between strangers to prevent Sybil attacks and align incentives.

The trade-off: Bitcoin uses a lot of electricity. This is controversial. But it's the cost of securing a monetary system without relying on governments or banks. Whether that trade-off is worth it is for society to decide.

For now, understand the mechanism: Proof-of-Work turns computation into authority. Temporary, rotating, earned authority.

And maybe you can start to see the general pattern: you need some “resource” to prove effort and earn others’ respect, then you need some rules so you cannot really do whatever you want with the earned temporal power, and finally you need incentives so people want to play by these rules.

Doesn’t this sound familiar? Doesn’t this sound similar to a democracy? Or at least to a voting system? How can this be democratic if not everyone has access to great computational resources or cheap electricity? We will dive deeper into the implications of all this later in the book.

10.12 What is a Blockchain, Really?

You can already actually know what all these fancy names are: Bitcoin, Ethereum, Solana, Cardano, etc.

For now I want to state that blockchain networks, cryptocurrency, or crypto are all horrible names. They hide the real nature of the technology.

But it is the name which has stuck.

These are better names for talking about this technology: - **Consensus-based database networks.**

When someone says “blockchain,” you can now understand that they are talking about a “consensus-based database network” where participants agree on the state of a shared database in a network.

In Bitcoin, the consensus mechanism is Proof-of-Work, but in practice, it can be ANYTHING—even a centralized consensus. An authoritarian consensus, an oligarchic consensus based on plutocracy where only the rich can write the next blocks. Etc.

Bitcoin users value the ability to compute hash computations really quickly as a difficult resource to get, which makes their rules impossible to break.

Ethereum started this way too and then changed their consensus for reasons we will explore later.

Finally, as humans we can just agree on distributed database states, and some of us think that is a good way to represent money.

Can we even write the laws on this database in such a way that no corrupted government can change them? Can we create a perfect voting mechanism that actually represents every citizen without tampered elections? How should we define the “effort” in the consensus for that? Should we even give so much power to the people if this is possible? Are we better in the hands of technocrats?

I would like the reader to feel empowered. Having the ability to coordinate with anyone about what constitutes valid data, valid information, eventually if you like, valid truth, is an incredibly complex and useful technological progress so we can better coordinate as a species.

And as uncle Ben said to Peter Parker: “With great power comes great responsibility.” We must study, we must understand, we must be careful. Slowly but surely build things that matter and

are useful for all of us.

For example, you can understand a casino as a coordination and as a consensus. People play consensual games in a casino and bet money. You can make your database represent a casino, set some rules, and allow anyone in the world to “gamble fairly” through the internet.

These casino-like applications are being the main thing that normal people find attractive and disgusting about “crypto”.

Please, do not get blinded by the fancy names, do not get blinded by the people who use these beautiful database structures to scam others or enhance their gambling addictions.

There is a lot of potential for good and species’ growth here. Like the one Bitcoin has, which I will explain later.

For now realize this: databases save data, you interpret the data, you and your neighbors give it meaning. Go ahead, coordinate with them, agree on truths without giving anyone excessive access to define the rules.

We will talk deeper about the implications of all this later in the book. But for now, it is essential to really understand what you are gaining and what actually all this technological progress is about.

10.13 A Note on Satoshi’s Thinking

I don’t know if Satoshi Nakamoto (whoever he or she is or they are) thought of it this way while designing Bitcoin. Maybe he did, maybe he approached it differently.

But this is how the author of this book makes intuitive sense of the natural progression: We need coordination, so let’s borrow centralization temporarily. We need fairness, so let’s rotate leadership. We need Sybil-resistance, so let’s require proof of effort. We need incentives, so let’s reward the workers.

The pieces fit together elegantly.

Key Insight: To solve the distributed database problem, Bitcoin borrows centralization temporarily—giving one person at a time the right to write and order everyone’s next transactions. But instead of picking leaders through voting (vulnerable to Sybil attacks) or random selection (also vulnerable), Bitcoin requires proof of computational effort. Whoever solves a difficult puzzle first earns the right to write the next block and receives a reward. This creates a rotating, temporary leadership where authority is earned through work, not granted by trust. Proof-of-Work aligns incentives: it’s more profitable to play honestly than to cheat.

Next, we’ll explore how these blocks are chained together over time, creating the blockchain structure that makes history tamper-evident, and we will keep diving deeper into important details and implications of this design.

11

Chapter 11: The Blockchain - Chain-ing History Together

What happens when two miners solve the puzzle at the same time?

At the end of Chapter 10, we left a critical question unanswered.

We've established who gets to write to the database (miners who solve puzzles), when they write (every 10 minutes), and why they follow rules (incentives align). But we haven't addressed a fundamental edge case: **what happens when two miners solve the puzzle at the exact same time?**

11.1 The Simultaneous Solution Problem

Imagine we all have synchronized databases showing the same balances at Block 100. (Remember, Block 100 just means we've agreed 100 times on the next state of the database—in Bitcoin's case, each state transition involves subtracting some balances and adding others based on transactions.)

Current state (Block 100):

- Alice: 50 coins
- Bob: 30 coins
- Carol: 20 coins

Now picture two miners, Miner A and Miner B, both solving the Proof-of-Work puzzle at nearly the same time. They both found valid solutions, they both did the work, and they both deserve the reward.

Miner A shares with the network Block 101a:

Block 101a:

- Transaction: Miner A receives a reward as per incentive rules, let's say 10 coins
- New balances: Alice: 50, Bob: 30, Carol: 20, Miner A: 10

Miner B shares with the network Block 101b:

Block 101b:

- Transaction: Miner B receives a reward as per incentive rules, let's say 10 coins
- New balances: Alice: 50, Bob: 30, Carol: 20, Miner B: 10

Both blocks are valid—both miners followed the rules, and both completed the Proof-of-Work.

Now the nodes have to decide: which block should they accept?

Some nodes receive Block 101a first and update their databases accordingly, while other nodes receive Block 101b first and update to that version instead. **The databases are now desynchronized.** Half the network thinks one miner received the reward, and the other half thinks a different miner did.

11.2 The Impossible Choice

Here's the dilemma we face:

If we choose Block 101a: - We reject Miner B's work, even though he was honest and did the work. - We break the consensus rule that says "whoever solves the puzzle gets to write." - Why would anyone participate if their honest work can be rejected arbitrarily?

If we choose Block 101b: - We reject Miner A's work, even though he was also honest. - Same problem—we're breaking the rules.

If we accept both: - The database state becomes inconsistent. - Half the network has different balances than the other half. - The entire point of consensus—everyone agreeing on the same state—is lost.

If we let the database desynchronize: - The system becomes useless because different people see different truths, different data. Never forget: all is just data, and what matters is how we interpret it. - If we're interpreting that data as money, it becomes worthless if we can't agree on who has what.

If we break the rules: - The consensus is broken. - The very rules that make people choose the system and find it valuable are violated. - The system becomes valueless.

This is a real problem—not just because of the potential for a malicious actor to rewrite history (we'll get to that), but also because honest participants creating valid blocks simultaneously can cause the network to diverge and lose synchronization.

11.3 The Elegant Solution: Let Them Compete

Both miners proved they did the work, and we can't choose one without being unfair.

So here's Bitcoin's answer: **Don't choose yet. Let them compete for one more round. Let's see who can sustain their effort.**

The rule: When two valid blocks appear at the same time, don't reject either immediately. Instead, see which one gets extended first.

How it works:

1. Some nodes accept Block 101a, others accept Block 101b.
2. Miners start working on the next puzzle (Block 102).
3. Some miners build on top of Block 101a, others build on top of Block 101b. Remember that the puzzle depends on the previous block's data, so they must choose one to build on.
4. Whichever block gets extended first (has another block built on top of it) wins.

Example:

```

    +- Block 101a (Miner A) <- Miner C starts building here
Block 100 ++
    +- Block 101b (Miner B) <- Miner D starts building here

```

A few minutes later, Miner C solves Block 102a (building on Block 101a):

```

    +- Block 101a -> Block 102a (this "chain" is now longer)
Block 100 ++
    +- Block 101b (this chain is shorter)

```

Now the decision is clear: the chain with Block 101a is longer, which simply means more effort has been put into solving puzzles to create it. Following our consensus of effort, we should pick that one.

All nodes switch to the longer chain, and Block 101b is abandoned. Miner B's work is discarded, but that's okay—the decision was made by playing by the very same consensus of effort. In Bitcoin's case, that's the consensus that quick hash computations represent a valid feat of effort.

The transactions from Block 101b that weren't included in Block 101a go back to the “mempool” (like a waiting room for transactions) and will be included in future blocks if they're still valid.

Okay cool, we solved it! We can now move on and keep synchronizing the database even when there are ties.

But what if they tie again, and again, and again? In general terms, what if the consensus consistently gives us two valid winners worthy of the right to write to the database? What if someone tries to trick us sending fake blocks in the future that look like they have more work on them?

If we want to create a robust system, we can't just hope for the best and assume this will never happen.

11.4 For How Long Do We Keep Doing This?

The answer isn't a general one—it's specific to the consensus mechanism you use in your consensus-based database network.

Remember, I think “blockchain networks” is a really bad name, so from now on in this book, I'll call them consensus-based database networks, or just consensus database networks, or even just CDN (Consensus Database Network).

The answer for Bitcoin's consensus:

The chain that accumulates the most Proof-of-Work wins. Since the puzzle miners are solving is truly random, eventually one chain will get ahead—and that can only be caused by the fact that those miners have put greater amounts of effort into it. The longest chain represents the most computational effort, and that's the one everyone accepts.

This is called the “longest chain rule.”

The explanation of why one chain eventually gets ahead has been simplified here. If you're curious, Satoshi Nakamoto's original whitepaper has a more formal explanation of why this works.

When there are multiple competing versions of the database, nodes follow this simple rule: **Accept the longest valid chain (consensus-iteration).**

Since we now have a new rule to account for, we have to describe it with new data. Now we need to store in our database not only people's balances but also how many steps of effort have been put into the database.

A naive approach would be to simply store a counter—a small number of bits that counts how many iterations of effort (blocks) have been added. But this is easily hacked.

How do you track iterations in a way that can never be faked? You can't simply assume the next valid block will be Block 101, because what if someone comes along and says, "Hey, I found a valid database where no one has double spent, all transactions are valid, and it says Block 102"? It might be that someone actually put more effort into the database and found 2 more blocks while you weren't paying attention.

But it could also be that 2 miners found a valid solution at the same time, and one was honest saying "the next block is 101," while the other was cheating saying "the next block is 102." If people see 102, they'll think more effort was put in there, and they'll all accept it.

The second miner has just bypassed the system and tricked everyone into thinking more effort was invested when it really wasn't.

Why was he able to achieve this? Because there's no link between each step we take in the database—anyone can just claim whatever they want about the next number.

We need a more clever way of representing which iteration we're on, a way that can't be cheated. A way in which no one can reinvent either the future or the past data of our database. A way in which no one can say, "We're suddenly in the future, guys" (higher iterations).

And here is where we finally introduce the blockchain data structure—a data structure that makes it impossible to fake time.

But before giving you this last core piece of the puzzle, let me clarify something important: Finality.

11.5 Understanding Finality

Finality is a technical term we use to define how much time has to pass before an iteration of our database is considered final—meaning it can never be changed again by anyone, ever.

The time to finality depends on the consensus you're using in your consensus-based database network. In Bitcoin, finality is probabilistic: the more blocks built on top of a given block, the more secure that block becomes.

After 6 blocks (about 1 hour), the probability of rewriting that block becomes astronomically low, so for practical purposes, we consider it final.

In other CDNs, like Ethereum, finality is achieved through different mechanisms. In Ethereum's case, it takes about 12-15 minutes to consider a block final.

This means that in a CDN, once finality is reached, no one will ever be able to change that data—even if they have 51% of the computational power in Bitcoin's case.

The property of finality is also enabled by the blockchain data structure, and different CDNs have different finality times.

11.6 Now, What Is a Blockchain, Actually?

It's simply a very clever way of storing which iteration of this never-ending tale of writing to our common database we're on—in a way that guarantees no one can cheat about it.

As you can see, the blockchain is actually just another piece of the entire system we're building up. It's a piece of engineering that fits into our machine.

Imagine calling automobiles “combustion engine transportation machines.” The engine is just a piece of the entire transportation system. What matters is its usage, which is why we use the word “automobile” (auto: by itself, mobile: moving) rather than “combustion engine transportation machine”—that would be too technical and not very useful.

The term “blockchain network” has a similar problem: it's too technical and not very descriptive. The industry has been using “blockchain network” for too long. At the very least, I prefer to say: A blockchain network is a consensus-based database network that uses the blockchain data structure to prove its history.

Being honest with myself, “consensus-based database network” is also too technical for most people. But at least it describes what these systems actually do, which is more important.

Normal people understand the word “data” pretty well—after reading this book, you understand it even better. People also understand “synchronization” and “decentralization”. So here I propose calling this entire class of technology **Decentralized Datasync Technology** instead of Blockchain Technology.

To be honest, you won't learn any new functionality from here on in this chapter.

The blockchain data structure is just a secure way of storing the counter—and also the transitions of data in our database's history—in a tamper-evident way. This ensures everyone knows which iteration we're on, which one comes next, which one came before, and that no one can cheat about any of it.

During the rest of the chapter I'll explain intuitively how this works, but this is more of a computer science topic than an “understand the usage of new technologies” one—which is what this book tries to focus on mainly.

I'm trying to explain as few technical details as possible while still giving you a true understanding of how these systems work.

11.7 Optional: For the Curious (You Can Skip This)

If you're more interested in the societal and practical implications of consensus-based database networks than in the computer science behind them, feel free to skip the next chapter and continue in chapter 13.

The following chapter dives into the technical details of how the blockchain data structure works—it's fascinating, but not essential for understanding the bigger picture of what these systems enable.

If you stay, we'll explore data structures, memory, pointers, linked lists, and finally put it all together to see exactly how a blockchain prevents cheating.

-> Continue reading if you're curious about the technical details, or skip to Chapter 13 for the next big topic.

Key Insight: When two miners solve puzzles simultaneously, we can't choose one without breaking consensus rules. The solution: let both chains compete, and accept whichever gets extended first (the longest chain rule). To prevent cheating about which iteration we're on, we use the blockchain data structure—a tamper-evident way of storing history. This enables a system where the past cannot be silently rewritten, and everyone can verify they're on the same database iteration. We call this entire class of technology **Decentralized Datasync Technology**—systems that synchronize data across thousands of computers without central control.

Next, we'll explore what these systems enable at a societal level—how they change power dynamics, enable new forms of coordination, and why this matters for the future. The foundation is complete. Now let's understand the implications.

12

Chapter 12: The Blockchain Data Structure (Technical Deep-Dive)

For the curious: How does blockchain actually prevent cheating?

In Chapter 11, we learned that when two miners solve puzzles simultaneously, we let them compete and accept the longest chain. We also learned that we need a clever way to store which iteration we're on—a way that can't be cheated.

This chapter is a technical deep-dive into how the blockchain data structure works. If you come from Chapter 11, welcome! If you're coming from another chapter—you might want to read all until Chapter 11 first for context.

12.1 How Does the Blockchain Data Structure Actually Work?

So we can finally explain what a blockchain is.

But first, we need to understand a few concepts from computer science. Don't worry—I'll keep it simple and build from what you already know.

12.1.1 What Is a Data Structure? And What Is a Computer's Memory?

Remember from Chapter 2 when we talked about **standards**? ASCII is a standard for representing letters with bits, and JPEG is a standard for representing images. **A data structure is like a standard, but more complex**—it defines how we store large amounts of data that represent complex ideas.

ASCII represents simple things like letters, but what if you want to represent something more complex, like a car? You'd need information about: - Make and model. - Year. - Color. - Price. - Owner.

A **data structure** defines how to organize all this information in memory so a computer can store and retrieve it efficiently.

Where is this data stored? In the computer's memory—which, as we know from Chapter 1, is just lots of transistors representing bits (0s and 1s).

You can think of memory, physically, as a rectangle made of metal with lots of tiny squares carved into it. If we zoom in conceptually, we can imagine it as a grid:

Memory Grid (simplified visualization):

0	1	1	0	0	1	0	1	← Row 0 (address 0)
1	0	1	1	0	0	1	0	← Row 1 (address 1)
0	0	1	1	1	0	0	0	← Row 2 (address 2)
1	1	0	1	0	1	1	1	← Row 3 (address 3)

Each row has an **address** (like a street address) so the computer knows where to find data. These addresses are built primarily at the hardware level, not the software level.

When you store information, you're essentially making electricity pass through the cables connected to those rows, setting these bits to specific patterns. When you retrieve information, you look up the address and read the bit pattern.

A data structure defines how to organize these bits in memory to represent complex information efficiently.

Now, inside each row (address), we can store different pieces of data—and let's call each row a “variable” because its content can vary. Since we're humans who like to interpret things, let's read whatever data is in the first row as a letter according to ASCII.

If we let electricity selectively pass through the first row, setting its bits to specific values, and we read it as ASCII, we can spell words. Here is the word “SAND” in memory:

Memory Grid storing "SAND":

0	1	0	1	0	0	1	1	← Row 0: 'S' (ASCII 83 = 01010011)
0	1	0	0	0	0	0	1	← Row 1: 'A' (ASCII 65 = 01000001)
0	1	0	0	1	1	1	0	← Row 2: 'N' (ASCII 78 = 01001110)
0	1	0	0	0	1	0	0	← Row 3: 'D' (ASCII 68 = 01000100)

Note that the physical image I've described is just a conceptual simplification. In reality, computer memory is far more complex, but this mental model helps us understand how data structures work accurately.

12.1.2 Pointers: References to Other Locations

Here's a powerful idea: **What if one piece of data points to another piece of data?**

Imagine you store a car's information starting at address 0. We can use a simple pointer system where the first piece of information tells us where the actual car data begins.

Let's say we as humans agree that: - Row 0 contains a pointer (an address) to where the car information starts. - When we follow that pointer, Row 1 contains the car type. - Row 2 contains

the country where it was made.

Memory Grid with pointers:

0	0	0	0	0	0	0	1	← Row 0 (address 0): POINTER to address 1
0	1	0	1	0	1	0	0	← Row 1 (address 1): Car type (Toyota)
0	1	0	1	0	0	1	1	← Row 2 (address 2): Country (USA)
0	0	0	0	0	0	0	0	← Row 3 (address 3): Empty

A pointer is just a number in memory that tells you where to find other data in memory.

Why is this useful? Because you can create relationships between pieces of data—one piece can “reference” another, be linked with another, chained with another. Do you see where this is going?

12.1.3 Linked Lists: Chaining Data Together

Now combine these ideas: **What if every piece of data contains a pointer to the next piece of data?**

In the car example, imagine that after the car data, we have another row of memory with a pointer to the data of another car the owner has. This creates a **linked list**—a chain of data where each element points to the next one.

Address 0	Address 3	Address 6
Car Data: A	Car Data: B	Car Data: C
Next: (3)	→ Next: (6)	→ Next: NULL

In this case, we’re representing in a computer which cars someone owns, chaining them together.

To read the list: 1. Start at address 0, read Car Data A. 2. Follow the pointer to address 3, read Car Data B. 3. Follow the pointer to address 6, read Car Data C. 4. NULL means “end of list.”

Linked lists are made out of pointers, and they let you chain pieces of data together in sequence.

12.1.4 Hash Functions (Review)

We’ve talked about hash functions before (Chapter 7 and Chapter 10).

A hash function is an algorithm that takes any input and produces a unique, fixed-size output:

```
hash("Hello") = d3a1f2
hash("Hello!") = 9f4e7b (completely different)
```

Properties:

- **One-way:** Can't reverse it (can't get "Hello" from d3a1f2).
- **Deterministic:** Same input always gives same output.
- **Unique (with nuances):** Different inputs produce different outputs.
- **Sensitive:** Change input even slightly, output changes completely.

Hash functions create **unique fingerprints** for data. A safe hash function must output large enough values to avoid the statistical chance of two inputs having the same output—but don't worry about these details. You don't need them to understand how a blockchain is built. You just need to remember the properties of hash functions.

12.2 Putting It All Together: The Blockchain Data Structure

A blockchain is a type of linked list where each element is a block, and each block contains:

1. **Data** (transactions).
2. **A pointer to the previous block** (but not a memory address—instead, the hash of the previous block).

Block 1	Block 2	Block 3
Transactions	Transactions	Transactions
Hash of Block 0	Hash of Block 1	Hash of Block 2
PoW solution	PoW solution	PoW solution

Each block contains the hash (unique fingerprint) of the previous block. This creates a chain.

Why use a hash instead of a memory address? Because Bitcoin is distributed—thousands of computers each have their own copy of the blockchain. Memory addresses are local to one computer, but a hash is the same everywhere. Everyone can compute the same hash for their local copy of the previous block and verify that the chain is correct.

12.3 Why This Structure Matters: Tamper-Evident History

Here's the magic property of this structure: **If you change any block, you break the entire chain.** It is not a pointer, but it works similarly. I added the concepts of pointers and linked lists because they help with visual understanding.

Remember hash functions? Change the input even slightly, and the hash changes completely.

If someone tries to change Block 100 (to steal coins, rewrite history, etc.), its hash changes:

```
Block 100 (original): hash = 0000abc123...
Block 100 (modified): hash = 0000xyz789... (completely different!)
```

But Block 101 contains the hash of Block 100. If Block 100's hash changes, Block 101 now points to an invalid hash—Block 101 breaks.

And since Block 102 points to Block 101, it also breaks.

And Block 103... and 104... and every block after that.

Changing one block breaks the entire chain.

Visual:

Original chain:

Block 99 → Block 100 (hash: abc123) → Block 101 (points to abc123) → Block 102...

Modified chain:

Block 99 → Block 100 (hash: xyz789) → Block 101 (points to abc123??) → BREAKS
 ↑
 Block 101 expects hash abc123,
 but Block 100 now has hash xyz789.
 Chain is invalid.

This is the tamper-evident property. You can't silently change history. Any modification is immediately visible because the chain breaks.

12.4 But Can't You Just Recompute All the Hashes?

Smart question. What if the attacker doesn't just change Block 100, but also recomputes the hashes for all subsequent blocks?

Step 1: Change Block 100
 Step 2: Recalculate hash of Block 100
 Step 3: Update Block 101 to point to new hash
 Step 4: Recalculate hash of Block 101
 Step 5: Update Block 102... and so on

Yes, you could do this. But here's the catch: **each block requires Proof-of-Work.**

Remember, to create a valid block, you must solve the computational puzzle (find a hash starting with many zeros). Across the entire network this takes, on average, about **10 minutes** of massive computational effort.

So to rewrite history: - Change Block 100 → Must redo Proof-of-Work (~10 minutes). - Fix Block 101 → Must redo Proof-of-Work (~10 minutes). - Fix Block 102 → Must redo Proof-of-Work (~10 minutes). - Fix Block 103... 104... 105...

If you want to rewrite 6 blocks, you need ~1 hour of computational work.

And while you're doing this, the honest network keeps moving forward, adding new blocks.

The race: - You: Trying to rewrite the past (starting from Block 100). - Honest network: Building the future (Block 106, 107, 108...).

If the honest network has more computational power than you, they'll always be ahead. Your forked chain will always be shorter.

And remember the rule from Chapter 11: the longest chain wins.

12.5 The Longest Chain Rule (Refreshser)

We introduced this in Chapter 11, but now you understand why it's so powerful.

When there are multiple versions of the blockchain, nodes follow a simple rule:

Accept the longest valid chain.

Why longest? Because the longest chain represents the most accumulated Proof-of-Work—the most computational effort invested.

Example:

```
Honest chain: Block 1 → 2 → 3 → 4 → 5 → 6 → 7 (7 blocks, most PoW)
Attacker chain: Block 1 → 2 → 3' → 4' → 5' → 6' (6 blocks, less PoW)
```

Nodes choose: Honest chain (it's longer)

The attacker's chain is rejected—not because it's “evil,” but because it has less Proof-of-Work.

This means: To successfully rewrite history, you need to: 1. Rewrite the past blocks. 2. Catch up to the current block height. 3. **Get ahead of the honest chain** (so yours becomes the longest).

If the honest network controls 51% or more of the computational power, the attacker can never catch up.

The deeper a block is buried (more blocks built on top of it), the harder it is to rewrite.

This is why people wait for “6 confirmations” before considering a Bitcoin transaction final. After 6 blocks (~1 hour), rewriting history becomes exponentially expensive.

12.6 The Anti-Gaslight Machine

This is why blockchain is sometimes called an “anti-gaslight” structure.

Gaslighting is when someone makes you doubt reality by denying facts repeatedly until you assume you are the crazy one and accept the lie.

Without blockchain:

```
Corrupt Authority: "Alice never sent Bob 10 coins."
Bob: "Yes she did! I have proof!"
Corrupt Authority: [deletes the record] "Show me the proof."
Bob: "...I can't. You control the database."
Result: Gaslighting succeeds.
```

With blockchain:

```
Corrupt Miner: "Alice never sent Bob 10 coins." [tries to change Block 100]
Bob: "Yes she did! Look at Block 100 in my copy of the blockchain."
Corrupt Miner: [changes Block 100] "My version says otherwise."
Bob: "Your Block 101 points to the wrong hash. Your chain is invalid."
Everyone else: "Bob is right. We reject your modified chain."
Result: Gaslighting fails. History is preserved.
```

Everyone has a copy of the chain. If one person tries to rewrite history, everyone else notices because the hashes don't match.

This is the power of distributed, tamper-evident history.

12.7 The 51% Attack

We mentioned in Chapter 10 that if someone controls 51% of the computational power, they have more chances to write blocks.

Now we understand the full implication: **With 51% of the hash power, you can rewrite recent history.**

Here's how:

1. You make a transaction (Alice sends 10 BTC to Bob for a car).
2. Bob gives you the car after 1 confirmation.
3. Secretly, you start mining a parallel chain from before your transaction, where you send those 10 BTC to yourself instead.
4. Because you have 51% of the power, your chain eventually becomes longer.
5. You broadcast your longer chain. Nodes accept it (longest chain rule).
6. Bob's transaction is erased. You have the car AND your BTC back.

This is called a double-spend attack.

But notice: - You can only rewrite recent history (last few blocks). Rewriting deep history (100+ blocks) is exponentially expensive even with 51%. - You can only double-spend your own transactions. You can't steal others' coins (you don't have their private keys). - The attack costs enormous amounts of electricity and hardware. - If detected, the network's value crashes, and your hardware becomes worthless. - The attack is visible—everyone sees two competing chains.

This is why Bitcoin requires a 51% attack to be broken. But it is an irrational attack. The cost outweighs the benefit for internal actors. Only external actors would do it. For smaller CDNs, nation-states might be able to afford it, but for Bitcoin, it's prohibitively expensive.

12.8 The Blockchain Structure Summarized

Let's bring it all together:

- 1. A blockchain is a linked list where each block contains:** - Data (transactions). - Hash of the previous block (the “pointer”). - Proof-of-Work solution.
- 2. Blocks are tamper evident thanks to hashes:** - Each block contains the hash of the previous block. - Change one block → breaks all subsequent blocks.
- 3. Each block requires Proof-of-Work:** - Rewriting history requires redoing all the computational work. - This makes historical tampering exponentially expensive.
- 4. Longest chain wins:** - Nodes accept the chain with the most accumulated Proof-of-Work. - Attackers must outpace the honest network to succeed.
- 5. Deep history becomes immutable:** - The more blocks built on top, the safer the history. - 6+ blocks = effectively permanent (for most practical purposes).

This is the blockchain. Not just a “chain of blocks,” but a tamper-evident, distributed, append-only history that makes rewriting the past computationally infeasible in decentralized database networks.

12.9 Why This Matters

Throughout history, those who controlled records controlled the truth:

- Governments rewrote history books to erase inconvenient facts.
- Banks altered ledgers to steal or fake funds.
- Dictators destroyed archives to hide their crimes.

Blockchain inverts this power dynamic.

No single entity controls the history. Everyone has a copy. Tampering is visible. Rewriting requires overpowering the majority.

This is the anti-gaslight machine. A shared, verifiable, tamper-evident history that no one person controls.

Whether it's money (Bitcoin), programmable contracts (Ethereum), or any data we care about—blockchain provides a way to coordinate on truth without trusting any single authority.

Key Insight: A blockchain is a linked-list-like data structure where each block contains the hash of the previous block, creating a tamper-evident chain. Changing any block breaks all subsequent blocks. Since each block requires Proof-of-Work, rewriting history means redoing all that computational effort. The longest chain (most accumulated work) wins, making deep history effectively immutable. This creates an “anti-gaslight” ledger where the past cannot be silently rewritten. Combined with distribution (everyone has a copy), blockchain provides verifiable history without central control.

Now that you understand the technical details of how the blockchain data structure works, you're even more ready to explore the bigger picture—what these systems enable at a societal level, how they change power dynamics, and why this matters for the future.

13

Chapter 13: Ethereum - The Consensual Computing Machine

What if the database could run programs?

We've spent the last few chapters understanding Bitcoin: a distributed database that lets strangers agree on who owns how much without trusting any central authority.

But Bitcoin transactions are simple. Alice sends 10 BTC to Bob, Bob sends 5 BTC to Carol, and that's it—just value transfer. Subtract a number here, add it there. There's a bit more complexity under the hood, but at its core, Bitcoin is just executing additions and subtractions.

That on itself is a program, a very simple one that adds and subtracts balances. Simple and... boring.

What if the database could do more than just track balances? What if it could **run complex programs**?

That is the idea that sparked **Ethereum**—a distributed database that not only tracks balances but also stores programs. These programs are called **smart contracts**.

13.1 Bitcoin Recap: Simple Transactions

Bitcoin transactions are straightforward instructions:

Transaction:

- From: Alice's address
- To: Bob's address
- Amount: 10 BTC
- Signature: [Alice's signature proving she authorized this]

The network verifies: Does Alice have 10 BTC? Yes. Is the signature valid? Yes. Then update the database—subtract 10 from Alice, add 10 to Bob.

Simple. Clean. Works perfectly for money, or just value in the sense of gold. Some people argue with very good reasons that Bitcoin is not money in the sense of cash but value in the sense of gold, but that is another debate.

For now just notice what Bitcoin's simple scripting can't easily do. It can't easily say "send money to Bob, but only if some complex condition X happens." It can't hold money until a certain date

and then release it to multiple parties if certain complex conditions are met. It can't program complex logic like loans or voting systems.

Bitcoin moves value from A to B. That's what it was designed for.

Ethereum asked: What if we could program anything?

13.2 The Insight: Programmable Transactions

Vitalik Buterin (Ethereum's creator—this guy, unlike Satoshi, is known and alive) realized: **What if we could program complex logic on the database?**

What if we could program a loan? Or a will? Or even a voting system?

Not just “send X to Y,” but: - **IF** goods are delivered **THEN** pay seller **ELSE** refund buyer. - **IF** I'm inactive for 1 year **THEN** send my funds to my heirs. - **IF** it's the 1st of the month **THEN** deduct subscription fee. - **IF** I am a valid voter **THEN** allow me to store my vote for this election. - **IF** citizen misbehaves, **THEN** freeze its funds. - **IF** my political party is losing, **THEN** add fake votes to it.

This is a smart contract. Not a legal contract (no lawyers involved), and no smartness either, but a program that runs on a CDN and enforces rules automatically.

The smart part depends on who writes it. And the “contract” word might come from the fact that you need cryptographic signatures to interact with it—in the “real world” you sign contracts and they enforce things, and here you sign data with cryptography which allows someone to run code on a database, so it is kinda like a contract.

The name is a bit weird, but the idea is powerful. What if we could program anything on this database?

Let me show you what this means with real examples.

13.3 Smart Contracts: Consensual Logic On Code

13.3.1 Example 1: Loans with Collateral

You want to take a loan. With a smart contract, you can program it: at date X, if money not returned, the collateral goes to the lender. That's it.

No banks, no paperwork, no credit checks. Just code enforcing the agreement.

But wait **the program can only read data from the database**, therefore, how does the program know what the real-world data is, like the price of your collateral which might be, let's say, a Netflix stock?

Great question. Programs on the database need information from the outside world (like prices, time, delivery confirmations). Who gives this data to the database so the program can read it? Isn't that a centralized trusted party?

Well, it might be, but that is a topic for another time. For now, imagine it is possible to reliably put real data into the program in a verifiable and trust-minimized way with economic incentives and game theory as we have been doing.

If you are curious, research deeper what we call in the industry **oracles**—the most famous is Chainlink. The main idea of these oracle protocols is that they put data into the CDN from the real world, like price feeds, weather data, sports results, etc., in a verifiable and trust-minimized way.

13.3.2 Example 2: Will (Inheriting Bits)

You want your Ethereum assets to go to your children if something happens to you.

Traditional solution: Write a legal will, give your passwords to a lawyer (risky), hope everything works out.

Smart contract solution:

Contract: Digital Will

- IF I don't interact with this contract for 2 years
 - THEN send 50% of my ETH to Child A's address
 - AND send 50% of my ETH to Child B's address.
- ELSE IF I do interact (prove that I'm alive)
 - THEN reset the 2-year timer.

This is basically unstoppable. Even if you lose your private keys, even if you die, the contract executes automatically after the time period. No lawyers, no court, no one can block it.

13.3.3 Example 3: Subscription (Automatic Recurring Payments)

You subscribe to a service for \$10/month.

Traditional solution: Give them your credit card. Trust they won't overcharge. Hope you remember to cancel.

Smart contract solution:

Contract: Subscription Service

- User deposits \$120 (for 1 year).
- Every 30 days, contract sends \$10 to service provider.
- User can cancel anytime, contract refunds remaining balance.

You control when to cancel. The service provider can't take more than agreed. The rules are transparent and automatic.

And there are no fees to all these middlemen that make credit cards work on the internet—just the fees of running the program on the CDN (Consensus Database Network like Ethereum), which can be less. Traditional credit card processing fees range from **2–3% per transaction** for merchants, while transaction fees on decentralized datasync technologies depend on network congestion—ranging from fractions of a cent to a few dollars, often significantly cheaper than credit card fees for many use cases.

13.4 The Ethereum Virtual Machine (EVM): Everyone Runs the Same Programs

Here's the magic: **Every node in the Ethereum network runs these smart contracts.** Look, the previous phrase is full of weird words you would have not understood before reading this book. I hope you can see the progress you are making and the actual complexity on all these new decentralized datasync technologies.

Remember from Bitcoin: Every node has a copy of the database, and when a transaction happens, every node verifies it and updates their copy.

Ethereum adds: Every node also stores programs on the database. When someone interacts with a program, every node **runs the program** and computes the result.

This is the Ethereum Virtual Machine (EVM): A virtual computer that exists across thousands of real computers.

13.4.1 Addresses: Still Based on Asymmetric Keys

One quick note before we continue: Ethereum addresses work the same way as Bitcoin addresses. They're still based on asymmetric cryptography (public/private keys) with the same properties we learned in Chapter 7.

They look a bit different because of different hashing and encoding schemes, but the mechanism is the same: - Your private key → Your public key → Your address.

Examples: - Bitcoin address: 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa - Ethereum address: 0x742d35Cc6634C0532925a3b844Bc9e7595f0bEb

Different formats, same underlying cryptographic principles.

The addresses shown were made up for illustration.

13.4.2 How It Works:

1. You write a smart contract (in a programming language like Solidity).
2. You deploy it to Ethereum (it gets stored in the database).
3. Anyone can interact with it (send it transactions).
4. Every node runs the contract's code with your transaction as input.
5. Every node computes the same result (deterministic).
6. The database updates based on the result.

Key property: Deterministic computation.

Same input leads to same output. Always. On every computer.

If Alice sends the loan contract a “pay back loan” transaction, every node runs the contract code, sees the condition is met, and updates the database to return her collateral.

Everyone agrees on the computation, just like they agree on the balances.

You can't just not run a program, because it is stored in the database. If you try to censor a program, or run it differently, the next iteration of the database you produce (the next block) will be rejected by everyone else because your results (the data in the database) will not match theirs.

Hence, altering the code to censor programs would require altering the code that creates the consensus, so if you do so, you are basically creating a completely new network—one where you will be alone, and useless.

13.5 Why This Is Revolutionary Innovation: Unstoppable Programs

Think about normal programs. Facebook can delete your account. PayPal can freeze your funds. Amazon can change their terms of service.

Why? Because they control the servers running the code. They can modify it, turn it off, or block you anytime.

Smart contracts are different, by default: - No single company can control them. - Once deployed, they run forever. - No one can shut them down, not even the creator. - No one can change the rules without everyone seeing.

This is consensus on computation, not just on balances.

Note my wording, by default. Programs are flexible, so you can program a smart contract that is controlled by some company, or that can be changed by some authority. But that is a choice you make when you write the code and when you interact with the code.

The same way someone can write code that is controlled by a company, someone can just write that very same code but controlled by no one. The power to decide which program to use is in your hands.

Both programs will be running forever on the CDN, but you get to choose which one to interact with.

So here's the crucial part: Anyone can deploy their own program, and you choose which program to execute. You are not forced to use program X if you think it is abusing your data, charging you too much, etc.

A bank is lending money with too much interest rate? Anyone can create a new bank program with a lower interest rate so clients can lend at more favorable terms.

This enables competition at the code level. Transparent and auditable.

This enables so much possibilities...

13.6 What Could This Enable? Let Your Imagination Run

A man once said, if you can dream it, you can code it.

Think about the possibilities:

What if you had a decentralized social media where people could pay you directly to access your content instead of giving total control to a social media company and their opaque servers and algorithms?

What if anyone could see the algorithm of this social media? No hidden manipulation, no secret ranking systems. Transparent code that everyone can verify.

What if you could create a decentralized organization where every decision is voted on by members, and the votes automatically execute changes? No CEO that can override the will of the community.

What if artists could program royalties into their digital art, so every time it's resold, they automatically get a percentage—forever?

The possibilities are vast. Banks, insurance, voting systems, supply chains, identity management—anything that involves rules, agreements, and trust can potentially be programmed on a CDN.

Ethereum is a consensual computing machine. Thousands of people agreeing on what programs to run on their precious information, which can represent and mean anything, because we interpret it.

But there are catches.

13.7 The Downsides:

13.7.1 1. You Have to Pay for It

The execution cost of the program, at the end of the day, is electricity, remember. Every node runs the program, consuming computational power.

Some companies might allow you to use their centralized computers for free because they sell your data and “private” information—but in a decentralized computer, everyone has to pay their fair share of the electricity cost so the model is sustainable.

Furthermore, when you deploy a program, that literally occupies some transistors and they also run on electricity all around the world, so that has a cost too. If you want to put your bank code in the database so anyone can execute it, you will have to pay for the space it uses.

Is this expensive? Can anyone afford it? We will answer these questions later.

For now, just understand: **computation on CDNs costs money.** This is by design. These costs are usually called **gas** costs. Why gas? I do not know. Maybe because gas powers cars and computation powers computers? I don't care, this industry already has so many weird names, forgive me if I do not question this one.

By the way, there is an extra reason to pay the more you consume. Imagine you tell the network to run a program that never ends—voila, you hacked the network, now nothing else can execute because everyone is busy running your infinite loop. Well, that now becomes economically impossible. The more computation you use the more you pay, so to compute forever you would need to pay forever... infinite money? Impossible.

13.7.2 2. You can build anything! Wait... anything?

With all these decentralized datasync technologies, you can build:

- **Decentralized exchanges (DEXs):** Trade currencies with people around the world without a bank in between. It's like a stock exchange, but run by code instead of a corporation.
- **Decentralized banking:** Borrow and lend money without a traditional bank. Just smart contracts enforcing the terms for everyone, everywhere.
- **DAOs (Decentralized Autonomous Organizations):** Organizations run by code

and votes and not needing always a board of directors approval. Think of it like a company where shareholders automatically control stuff like budget through transparent voting. - **Identity systems:** Own your online identity across platforms, not controlled by Google or Facebook. - **Global gambling:** Program your unstoppable global casino to promote gambling worldwide without any government able to shut it down.

And many more new possibilities, these are just a few.

Some of these are useful. Some are hype. Some are experiments. Some are of devious ethics. But the **capability** is now real.

13.7.3 3. Code Can Have Bugs (bugs means errors in the code)

The code can get hacked if you do not code it properly. You leave your loan with a bug that can accidentally multiply by 10 what you owe? You are in trouble.

Even if the CDN works perfectly, the code you write on top of it can have bugs. So be careful which programs you run.

Some errors in the code (bugs) might remain dormant for a long time, until someone discovers them and exploits them. This has happened multiple times already, with millions of dollars lost—notably, The DAO hack in June 2016 resulted in the loss of about 3.6 million ETH (roughly \$60M at 2016 prices). Total crypto hacks in recent years have reached into the billions per year.

Fortunately, the security of the codes being used is improving year after year, but this risk is still present. The chances of it affecting you get reduced, but they will never be zero.

Personally, the author works precisely in this part of the industry: cybersecurity of CDNs and smart contracts. So I can tell you: this is a real risk, but it is being taken seriously by lots of professionals.

Slowly but surely, the chances of you losing your money because of these reasons will go to 0 in practice, and if it happens, insurance protocols will cover your losses, which are not really your fault.

It will feel like hiring a life insurance being 25 years old just in case a lightning hits you. It is very unlikely, but if it happens you want to be covered.

As of today when I'm writing this, December 16th, 2024, it is very risky, the author would not put all his savings/investments into any single cryptocurrency due to potential security issues.

13.7.4 4. The Devil Is In The Details I'm Hiding For Simplicity.

Some of the uses I've provoked your mind with are very complicated to integrate technically. Not impossible, but indeed way more complicated.

At least, with the knowledge you will get from this book, you will be better equipped to keep understanding them on your own, making your own critical, logical and informed questions about them while learning how they work.

13.8 Welcome to the Dark Side

As you might have noticed, I've described some use cases that are a bit... morally gray. Gambling, unregulated loans, corrupted elections, etc.

As with any new technology, there are both good and bad uses. Some people will use CDNs for innovation and liberty. Others might use it for scams, fraud, to evade legitimate regulations or to create illegitimate ones.

This is why you must understand the technology. To make sure you use it for “the betterment of the world” and not let someone else use it to abuse you.

This is like guns, cars, or the internet itself. Powerful tools that can be used for good or evil. We have the duty to care, to understand them at least at a certain level of detail, so we can make informed choices. This book is a clear effort in that direction.

13.9 Sum Up: Bitcoin Coordinates Value, Ethereum Coordinates Value And Computation

Let's zoom out and see what we've unlocked:

Bitcoin: - Distributed database. - Tracks balances (who has what money). - Consensus on simple rules (everyone agrees on ownership and simple transfers). - Simple transactions: “Send X to Y.”

Ethereum: - Distributed database + distributed computer. - Tracks balances AND runs programs. - Consensus on complex computation (everyone agrees on program outputs). - Complex logic: “IF condition THEN action.”

Both are coordination technologies. They let strangers agree on something without trusting a central authority.

Bitcoin: “We all agree Alice has 10 BTC.”

Ethereum: “We all agree this program should send funds to Bob because the condition was met.”

13.10 Story Time...

13.11 But Ethereum Started with Proof-of-Work. Then Something Changed...

When Ethereum launched on **July 30, 2015**, it used the same consensus mechanism as Bitcoin: **Proof-of-Work**. Miners solved computational puzzles, burned electricity, earned rewards.

But on **September 15, 2022**, Ethereum did something unprecedented: **It switched consensus mechanisms**. That event was called:

The Merge: Ethereum transitioned from Proof-of-Work to **Proof-of-Stake**.

No more mining. No more burning electricity. A completely different way to achieve consensus.

Why? One major reason: Proof-of-Stake consumes way less electricity (about 99.95% less).

Here's the key insight: This is consensus. If people (nodes) want a different algorithm, they can just agree and change it.

The community agreed this change was worth it. They voted with their participation. The switch happened. The network kept running.

They just said on an internet forum something like: At block number X, we will run a completely different code, okay?

And when that block came, they all did. Notice a very important detail: they ran different code, but the database state remained the same. All balances, all programs, everything stayed intact. Just the way consensus was achieved changed.

Consensus is social, remember? Technology enables it, but humans decide the rules.

If you're curious about the technical details of how Proof-of-Stake works, you can research it deeper. But for our purposes, just understand: it's another consensus mechanism, another way to coordinate, with another unique set of trade-offs.

For example, if you want to censor Bitcoin you have to control 51% of the computing power of the network. In Ethereum, computing power now matters nothing. What matters is... its very same native currency, Ether.

The important take on all this? The networks can evolve. Consensus can change. As long as all the participants agree.

But, thanks to that beautiful data structure, the blockchain, all the history is preserved. The past is immutable, but the future can be shaped by consensus. You can literally see all the changes that happened over time, who proposed them, when, and how the community agreed to execute them. No one can censor how the past happened, no one can rewrite history, but the future is open to collective human choice.

Now, what happens if not everyone agrees? What if the community splits? This has already happened multiple times in multiple CDNs. Let's explore that in the next chapter.

Key Insight: Bitcoin coordinates value, Ethereum coordinates value and computation. Smart contracts are programs that anyone can deploy (deploy on = save in) on the database, executed by every node with deterministic logic. You choose which programs to interact with—creating transparent competition at the code level. This enables loans, wills, subscriptions, voting, and countless applications without middlemen. But there are trade-offs: computation costs money (gas), code can have bugs, and powerful tools can serve both good and evil purposes. Programs are unstoppable by default, but developers can choose to make them controllable. When the Ethereum community decided to switch from Proof-of-Work to Proof-of-Stake, they did—because consensus is ultimately social. Networks can evolve when participants agree. But what happens when they don't agree?

Next, we'll explore what happens when consensus splits—when the community disagrees so fundamentally that the network splits into two separate realities. This will deepen our understanding of what consensus really means and why it's ultimately a human choice, not just a technical mechanism.

14

Chapter 14: When Consensus Splits - The Nature of Agreement

What happens when not everyone agrees?

We've learned that consensus is social. Ethereum switched from Proof-of-Work to Proof-of-Stake because the community agreed, and the network evolved accordingly.

But here's the question: **What happens when the community doesn't agree?**

What if half the participants want to go one direction, and the other half wants to go another?

This has happened. Multiple times.

14.1 The Fundamental Question

Remember what we learned: CDNs work because everyone runs the same code and agrees on the same rules.

But code is just software. Anyone can copy it. Anyone can modify it. Anyone can run their own version.

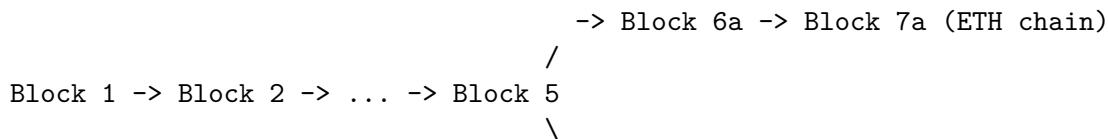
When a significant portion of a community disagrees on which code to run, we get what we call a **fork**. Think of it as a bifurcation in the road—two cars driving towards that split can choose different paths, and from that point on, they're traveling on different roads, heading in different directions towards different worlds, poetically speaking.

At a technical level, nodes simply start rejecting blocks that don't follow their version of the rules. Miners start mining on the version they prefer. Users start transacting on the version they trust. The result? Different databases with different data for every group running different software.

A blockchain, as you know from chapter 12, can be visualized as a chain of blocks:

Block 1 → Block 2 → Block 3 → Block 4 → Block 5...

What happens when the next state of the database differs between groups? Well, our cute drawing of the blockchain has to be drawn with a split in two—it forks, bifurcates:



-> Block 6b -> Block 7b (ETC chain)

This is what happens now:

So what stops the network from splitting into multiple incompatible versions?

The short answer: Nothing.

The long answer: Let me tell you two stories.

14.2 Story 1: The Ethereum Fork - Code Is Law vs. Protect The Ecosystem

In 2016, something unprecedented happened on Ethereum.

14.2.1 The DAO: A \$150 Million Experiment

Someone created a smart contract (program) called “The DAO” (Decentralized Autonomous Organization). It was essentially a venture capital fund designed to gather money from investors, but run entirely by code.

People sent money (ETH) to this contract, and token holders voted on which projects to fund. No CEO, no board of directors—just code enforcing rules.

It raised \$150 million worth of ETH. At the time, that was about 14% of all Ethereum in existence.

Everyone was excited. This was the future! Decentralized organizations!

Then, someone found an error in the code—a bug.

14.2.2 The Hack: \$50 Million Stolen

On June 17th, 2016, an attacker exploited a vulnerability in The DAO’s code.

The bug allowed them to repeatedly withdraw funds without the contract properly updating the balance—like an ATM that gives you money but forgets to subtract it from your account.

\$50 million worth of ETH was drained.

The attacker didn’t “hack” Ethereum. They didn’t change any consensus rules. They just found a flaw in the smart contract code and exploited it—perfectly legal according to the code itself.

If you write code that sends me money but forget to put a limit on how much it can spend, well, that’s not technically my fault. The actual hack was more complex than this, but you get the idea of a code error:

```
code IAmAliceAndHeIsNotPayingBack() {
    sendMoneyBackToAlice(100$);
    moneyIOweToAlice = -0$;
}
// This pseudo-code subtracts nothing from the debt,
// allowing infinite withdrawals to Alice. Oh no!
```

In the DAO hack, the code did exactly what it was programmed to do. The attacker just understood the code better than its creators and recognized an error in its programming.

Remember from the previous chapter: programs on the chain have their own rules, and if you want to override them, you can't do it without completely changing the state of the database—which leads to people having different states and rejecting each other's blocks.

14.2.3 The Dilemma: Two Incompatible Philosophies

The Ethereum community faced a choice, and both options had strong, internally consistent arguments:

Option 1: Code is law. Let it stand. - The attacker followed the code's rules (even if unintended). - Reversing this sets a dangerous precedent: it signals that breaking base consensus is okay if someone messed up programming on top of it. - Who decides what's a "legitimate" transaction vs. a "hack"? In this case it was clearly a hack, but what about future cases? These distinctions can get very tricky. - If we can reverse this, we can reverse anything.

Immutability matters more than the money because it signals we are honest to our consensus and therefore reliable.

Imagine a country with a very unstable legal system where laws can be changed retroactively and quickly. No one would want to do business there. What if one afternoon, out of the blue, your bank account is emptied because of a new law that says "all money in banks from X type of companies now belongs to the state"?

Option 2: This hurts the ecosystem. Fix it. - \$50M represents real people's money. - The attacker exploited a clear error in the code, not legitimate behavior. - If we don't fix this, trust in Ethereum collapses. What will people think? Will they understand the nuances between smart contract bugs and network-level agreements? Ethereum was just one year old at the time, and people were still learning what it was. - We can fork—rewrite history to undo the hack. - **Pragmatism matters more than ideology.**

Both sides had valid points. Both were internally consistent. Both reflected real values.

This wasn't a technical problem. It was a philosophical disagreement about what Ethereum should be.

Should Ethereum be a platform where **code is absolutely law**, even when that leads to theft? Or should it be a platform that **protects its users**, even if that means breaking immutability?

14.2.4 The Vote: 85% vs. 15%

The community held a vote (via a signaling mechanism—not binding, but indicative).

85% voted to fork — reverse the hack, give people their money back.

15% voted against — keep the chain as-is, honor "code is law."

The tension was palpable. Both sides felt they were fighting for the soul of Ethereum. Both sides believed they were right.

So Ethereum forked.

14.2.5 The Split: Two Etereums

At block 1,920,000, the network split into two separate chains:

Ethereum (ETH): The majority chain, the one still running as “the main one” today. It reversed the hack and returned the stolen funds. The community chose pragmatism under the promise of never doing it again—and so far they’ve kept that promise, even with major exchange hacks like the **February 2025 Bybit hack** where approximately 401,000 ETH (~\$1.5B) were stolen from the centralized exchange (not all was inside Ethereum but you get the idea).

Ethereum Classic (ETC): The minority chain. It kept the original history, honored immutability, and chose ideology.

Both are valid CDNs. Both are still running. Both have value. Both have communities.

Market values (which fluctuate constantly): - At the time of writing (early 2026), Ethereum’s market cap is in the **hundreds of billions of dollars**, while Ethereum Classic’s is in the **low single-digit billions**.

The market voted with its money. The majority won economically. But the minority still exists.

Now that you know a bit of the technical processes behind forks, let me explain what happened in practice:

At block 1,920,000, some nodes chose to follow the ETH chain, accepting a new block with a special transaction returning the hacked funds. Others chose to follow the ETC chain, rejecting that block and continuing with the original rules.

Miners split. Developers split. Users split. Exchanges had to decide which chain to call “Ethereum” and which to call “Ethereum Classic.”

The community was divided, but both could coexist.

14.2.6 Be Careful, The Past Can Haunt You

Remember the blockchain data structure? Its power can clearly be seen here. This isn’t just a story—**it’s recorded history**.

Go look at block 1,920,000 on both chains. You’ll see the fork. You can trace the stolen ETH. You can see where it went on each chain.

On Ethereum (ETH): The funds were returned.

On Ethereum Classic (ETC): The attacker kept them.

No one can hide this. No one can rewrite it. The current ETH blockchain preserves the disagreement forever. It will always display how the community broke consensus for one iteration, even if afterwards they resumed using the very same consensus rules.

As the newer generations would say: **This is the anti-gaslight machine in action.**

For anyone reading this who doesn’t know what gaslighting is: Gaslighting is a form of psychological manipulation in which a person or group makes someone question their own memory, perception, or sanity. For example, in a toxic relationship, one partner might repeatedly deny events that the other clearly remembers, making them doubt their own recollection and feel confused or crazy.

In this case, the blockchain prevents gaslighting by making the history of disagreements public and verifiable. No one can say “that never happened.” The data is there, forever—until the last node turns off, anyway. There are some nuances to that, but you get the idea. I encourage you again to research if you’re curious.

14.3 Story 2: Bitcoin vs. Bitcoin Cash - The Block Size War

In 2017, Bitcoin faced its own split.

14.3.1 The Problem: Bitcoin Is Slow

Bitcoin processes approximately 7 transactions per second. Visa processes around 24,000.

As Bitcoin grew popular, transaction fees skyrocketed and wait times increased. It was getting expensive and slow.

Why so slow? Because every node processes every transaction, and Bitcoin’s protocol limits block size to 1 MB (megabyte).

A megabyte (MB) is 1 million bytes. A byte is 8 bits. So 1 MB = 8 million bits.

Block size is like the maximum amount of data you can change in the database in each iteration. Each change of balances (a transaction) writes and deletes some data, so there’s a limit to how many transactions can be processed at a time.

If you can only write 8 million bits per iteration, and each iteration takes 10 minutes (Bitcoin’s target), you’re limited in how many transactions you can fit.

14.3.2 The Debate: Bigger Blocks vs. Keep It Small (size matters)

Once again, the community split into two camps with valid but incompatible visions:

Team Big Blocks: - Just increase the block size. - Allow 8 MB blocks instead of 1 MB. - More transactions per block = faster, cheaper. - We need to scale now to compete with Visa. - Bitcoin should be usable for everyday payments.

Team Small Blocks: - Bigger blocks = fewer people can run nodes (because they’d need more storage and bandwidth to handle the increase in data). - Fewer people running nodes = more centralization. - Centralization defeats the purpose of Bitcoin. - Increase transaction capacity with Layer 2 solutions instead (like Lightning Network). Don’t worry, we’ll touch on what a Layer 2 solution is later in this book. - Bitcoin should be a settlement layer, not a payment network.

Again, both sides had valid points. Both reflected different values about what Bitcoin should be.

One side prioritized **accessibility and low fees**. The other prioritized **decentralization and security**.

14.3.3 The Fork: Bitcoin vs. Bitcoin Cash

The community couldn’t agree.

So on August 1st, 2017, Bitcoin forked:

Bitcoin (BTC): Kept 1 MB blocks. Focused on decentralization. Developed Layer 2 solutions.

Bitcoin Cash (BCH): Increased to 8 MB blocks (later 32 MB). Focused on transaction throughput.

Both still exist. Both have communities. Both have different visions for what “Bitcoin” means.

Market values (which fluctuate constantly): - At the time of writing (early 2026), Bitcoin’s market cap is in the **hundreds of billions of dollars** (approaching or exceeding a trillion at times), while Bitcoin Cash’s is in the **low billions**.

Again, a majority clearly won economically. But the minority chain survives.

14.4 The Pattern: Technology Enables, Humans Decide

Notice what’s happening in both stories:

14.4.1 1. Technology Enables The Fork

The code is open source. Anyone can copy it. Anyone can modify it. Anyone can run their own version.

Forking is trivially easy from a technical standpoint. Just change a few lines of code, announce it, and see who follows.

The blockchain data structure makes it easy to prove where the split happened. The open-source nature makes it easy to copy and modify.

14.4.2 2. Humans Decide The Outcome

But which fork has value? That’s not a technical question—it’s a social one.

People vote with their participation on the network and with their money: - Which nodes do miners point their hardware at? - Which chain do developers build on? - Which coin do exchanges list? - Which coin do users buy and hold? - Which fork do people value?

The answer determines which fork “wins” economically (though both can survive).

In both Ethereum and Bitcoin’s case, the majority chain won the economic battle decisively. But the minority chains didn’t disappear—they found their own communities who valued their principles.

14.4.3 3. Blockchain Makes Disagreement Auditable

In traditional systems, disagreements can be hidden, censored, or rewritten.

In CDNs, the disagreement is permanent.

Thanks to the immutability property of the blockchain data structure, you can see exactly when the fork happened. You can trace which users went which direction. You can verify the history yourself.

No one can say “that never happened.” The data is public.

This creates a permanent, auditable record of every major disagreement in the network’s history.

14.5 Forks Are Civil Wars, Kind Of

When communities disagree fundamentally, they can split.

At first, forks sound bad. “The network split! Isn’t that a failure?”

Not really. **Forks are proof that coordination is voluntary.**

You might think: “Wouldn’t it be better if everyone just agreed? Wouldn’t that make us stronger? We’re weaker divided—that seems bad.”

That’s a totally valid point of view. Here’s where ethics comes into play, and we realize that ethics aren’t objective.

Are forks bad or good? It depends on your values and the context in which they happened.

If you value unity and strength through scale, forks look like failures. The community is weaker divided.

If you value freedom and the ability to exit, forks look like successes. The minority has power.

If you value immutability above all, the ETH fork was a betrayal. If you value pragmatism and protecting users, it was a necessary intervention.

If you value accessibility, Bitcoin Cash’s bigger blocks make sense. If you value decentralization, Bitcoin’s small blocks make sense.

There is no objectively correct answer. This is philosophy, even politics if you will—not mathematics.

14.5.1 You Can’t Force Global Consensus

No one can make you run specific code. No one can force you to agree.

Well, if all the nodes are in one country, that country’s military might be able to... but most CDNs are global and distributed across countries that don’t necessarily like each other.

If 85% want to reverse a hack, the 15% can say “no” and keep the original chain alive.

If you disagree strongly enough, you can fork. And if enough others agree with you, your fork survives.

This is a technology of collective freedom. The freedom for any community to use the technology that better suits their interests.

14.5.1.1 How Many People Do You Need To Fork?

Just the 2 of us.

As long as you have a group—meaning at least 2 people—you can both run your own version of the code on the database and convince others to use it. As long as you have at least 2 people, you have consensus among yourselves and can run your own version of the code.

Is that economically sustainable? Do you two have enough money to run the computers even if no one is willing to pay to write data on your database? If yes, you can run your own version of

the code and create your own “digital society,” your own CDN. If not, well, you’d better find users willing to pay for your services.

Or threaten them, who knows? Never forget the dark sides of human nature. But don’t be too paranoid either. In balance we find virtue, Aristotle would say.

But who am I to tell you what to do? No one—the same way I’m no one to tell you what consensus to run on your computer.

14.5.2 History Of Disagreement Is Preserved

Governments rewrite history textbooks. Corporations delete embarrassing records. Centralized platforms ban dissenting voices.

CDNs, because they use blockchains, can’t do this.

The Ethereum/Ethereum Classic split is visible forever. Anyone can study it. Anyone can learn from it.

Future generations will see: - What was the disagreement about? - Who voted which way? - What were the arguments? - How did the market react?

No one can censor this. The blockchain makes disagreement permanent and auditable.

14.6 The Deeper Realization: From Two Societies To Many

If consensus can split into two societies (ETH/ETC, BTC/BCH), can it split into many?

Yes.

You could have: - Ethereum (pragmatic, majority chain). - Ethereum Classic (ideological, immutability-focused). - Ethereum [New Fork] (experimenting with different features). - And more...

Each fork is a mini-society with its own rules, its own community, its own values.

14.7 Enough Division—Why Not Unite?

Here’s a provoking thought: What if instead of splitting entirely, we created mini-societies that coordinate with a larger mega-society?

What if we had: - A main chain (slow, secure, expensive, global consensus). - Many side chains (fast, cheap, specialized, local consensus). - All coordinating together when needed.

Like the United States: - Federal government (slow, secure, final arbiter). - 50 state governments (fast, local, specialized). - Both working together.

Or something “slightly” different, like the European Union: - EU (coordination layer, shared rules). - 27 nations (sovereign, independent, specialized). - Both respecting each other.

This is Layer-2 scaling. And it’s the topic of our next chapter.

Very summed up and leaving details behind, for now:

Layer-2s are simply CDNs that communicate with another CDN, sharing data when needed but having their own rules and code to run on their own.

Instead of complete separation (like forks), Layer-2s maintain connection while allowing independence. Best of both worlds.

14.8 What This Means For Anyone

Forks teach us something profound:

Software consensus is not something you enforce. It's something you choose.

You can't force people to agree. You can only: - Make your case. - Write your code. - See who follows.

The network is, ultimately, the people who chose it. Code is just a tool. People decide which tool to use. If enough people disagree, the network splits. And that's not necessarily the end of the world, since both versions can coexist.

This is what decentralization really means: No single entity decides for everyone. Groups do, starting from the smallest one—2 people.

This is not individual freedom, but collective freedom. A technology that allows any collective to coordinate their information in the way they please.

14.9 The Anti-Gaslight Machine, Upgraded

We've been calling blockchain "the anti-gaslight machine" because it prevents rewriting history.

But now we see something deeper:

You can't prevent consensus from changing. People will always disagree. Forks will happen.

But you can prove when and how it changed.

If a majority tries to rewrite history, the minority can fork and preserve the original.

If a government tries to censor a transaction, users can fork and keep it visible.

If developers try to impose new rules, users can reject them and stay on the old code.

No one has absolute power. Everyone has the power to exit.

This is a fundamentally new social structure that enables instant voluntary data coordination with permanent records of disagreement.

The blockchain doesn't prevent conflict—it makes conflict visible, auditable, and survivable.

Key Insight: Forks happen when communities disagree fundamentally about values and direction. Are they good or bad? It depends on your values and context. The Ethereum fork (DAO hack) split over pragmatism vs. immutability. The Bitcoin fork (block size) split over accessibility vs. decentralization. Both sides had valid arguments—this is philosophy, not math. Technology makes forking easy (copy code, run your version), but humans decide the outcome (which chain

has value). The blockchain makes disagreement permanent and auditable—you can verify exactly when/how it happened. Forks prove coordination is voluntary: you can't force consensus, only choose it. With as few as 2 people, you can fork. The anti-gaslight machine upgraded: you can't prevent change, but you can prove when it changed and preserve alternatives. No one has absolute power—everyone has the power to exit. But what if instead of complete separation, we coordinate at multiple levels?

Next, we'll explore Layer 2 scaling solutions—mini-societies that coordinate with larger mega-societies. Like US states and federal government, or EU nations and the EU. Nested consensus, societies within societies, all working together. This is how CDNs scale without forcing everyone to split completely.

15

Chapter 15: A Society For You! A Society For Me! A Society For Everyone!

Can we have the best of all worlds?

In Chapter 14, we learned that when communities disagree, they can fork. Complete separation. Different chains, different communities, different values.

But what if we didn't have to choose between unity and independence?

What if we could have **many coordinating societies** instead of one global society or many completely separate ones?

This is the idea behind Layer 2s. I guess they are called this because they are usually drawn alongside the other blockchain, looking like they are forming layers:

Layer 2 Chain: ->->-> [] ->->-> [] ->->-> [] ->->-> [] ->->-> [] ->-> Block, Block

Layer 1 Chain: ->->-> [] ->->-> [] ->->-> [] ->->-> [] ->->-> [] ->-> Block

But before understanding Layer 2s better, it will help to understand a fundamental problem.

15.1 The Blockchain Trilemma (like a dilemma but with three options)

There's a famous problem in CDN design called **the blockchain trilemma**.

It states that you can only optimize for **2 out of 3** properties:

1. **Decentralization:** Anyone can participate (run a node, verify transactions...).
2. **Security:** The network can't be easily attacked or compromised.
3. **Scalability:** The network can handle lots of transactions quickly and cheaply.

You can pick any 2 and be very good at them, but not all 3.

Let's intuitively explain why.

15.1.1 Decentralization + Security = Slow (Bitcoin, Ethereum)

This is what Bitcoin and Ethereum chose.

Decentralization: - Anyone with a computer can run a node. - Blocks are small enough (1-2 MB) so many people can store and verify them. - Thousands of independent nodes operate worldwide.

Security: - To attack the network, you need a large amount of a difficult-to-obtain resource. - Extremely expensive and difficult.

Yet, this comes at a speed cost: - Every node processes every transaction. - Every node stores every block. - Every node must reach consensus with every other node.

There's a physical limit to how fast this can happen. If blocks come too quickly, some nodes won't be able to keep up with their smaller machines. If blocks are too large, only people with expensive hardware can participate, which leads to centralization and loses the security guarantees that decentralization provides.

The block time (how often new blocks are created) is chosen in the consensus rules. Bitcoin chose 10 minutes, Ethereum chose ~12 seconds. But you can't make it too fast without risking that only powerful computers can keep up.

Result: **~7 transactions/second (Bitcoin), ~15 transactions/second (Ethereum).**

Compare to other centralized payment systems like Visa: **~24,000 transactions/second.**

15.1.2 Scalability + Security = Centralized

What if we wanted to handle 24,000 transactions per second?

Option: Make blocks huge (like 1 GB instead of 1 MB).

Result: - Can fit way more transactions per block. - Much faster, much cheaper.

But the problem: - 1 GB blocks every 10 minutes = 144 GB per day = 52 TB per year. - Most people can't store that much data. - Most people can't download 1 GB blocks every 10 minutes. - **Only large corporations and data centers can run nodes.**

Result: **Centralization.** You've almost recreated a traditional database with extra steps.

Congratulations, instead of a traditional data dictatorship now you have a small oligarchy. I mean, technically it is actually more decentralized.

15.1.3 Decentralization + Scalability = Chaos (Insecure)

What if we wanted lots of people to participate (decentralization) AND process tons of transactions quickly (scalability)?

Option: Keep the network open to everyone, but make blocks huge and come very fast.

What happens: - Lots of transactions fit in each block (scalability check). - Anyone can still join (decentralization check). - But you have to process things very, very quickly.

The problem:

Not everyone has the hardware to keep up. Imagine you're processing block 10 while someone in Japan is already on block 12, another node in Brazil is still on block 11, and the rich guy with the fastest computer is on block 20.

The database becomes completely messed up and out of sync.

Furthermore, physical distances matter. Even at the speed of electricity, if blocks come too quickly, a node in Iceland and a node in Senegal can't stay coordinated. By the time Block 100 reaches Senegal, Iceland is already on Block 105. The network fragments geographically.

It's like a Michelin-star restaurant's kitchen where everyone literally runs around trying to coordinate at impossible speeds. Chefs would clash into each other, start shouting, get confused. Chaos.

Result: **Insecurity**. The network can't maintain consensus. Nodes disagree on the current state. The whole system breaks down.

15.2 Why Decentralized CDNs Are Inherently Slow

At least with modern algorithms, cryptography, and hardware availability: **Global consensus takes time**.

Remember how CDNs work: 1. People send transactions. 2. **Every node** receives and verifies it. 3. Another node (miner/validator) proposes a block with that transaction. 4. **Every node** verifies the entire block. 5. **Every node** updates their local copy of the database.

This is coordination at a global scale.

The more nodes you have (and therefore more decentralization), the more work that needs to happen. Every node must compute several things for every transaction.

If you want it secure—that is, working as expected—you can't rush it. If you want it decentralized, you can't limit who participates. But if you want it fast too... you have to sacrifice one of those.

15.3 Do We Need Slow Global Consensus For Everything Though?

What if we don't need global consensus for every single transaction?

What if we can run CDNs that sacrifice a certain degree of decentralization in exchange for speed, but only for certain use cases?

Think about real life:

- You don't need the federal government to approve every purchase you make at a local store.
- You don't need the UN to validate every contract between two people in the same city.
- You don't need every human on Earth to agree on what you and your friend do together.

Most coordination can be local. Only some coordination actually needs to be global.

So what if we applied this to CDNs?

15.4 Layer 2 Solutions: Mini-Societies Within A Mega-Society

The idea: Create smaller networks (Layer 2s) that sit “on top of or alongside” a main chain (Layer 1).

Layer 1 (The Base Layer): - The “federal government” or “EU level.” - Slow, expensive, maximally secure, global consensus. - Final arbiter when disputes arise. - Everyone trusts it, but you don't use it for every little thing.

Layer 2 (Scaling Layers): - The “state governments” or “member nations.” - Fast, cheap, specialized, local consensus. - Handles day-to-day transactions. - Periodically “settles” with Layer 1 for security.

Both working together.

Instead of processing every transaction on the main chain, we process most transactions on Layer 2s and only use Layer 1 when we need to create a checkpoint, settle, or resolve disputes.

Like running a tab at a bar. You don’t usually pay after every drink—you settle up at the end of the night.

15.4.1 Examples of Layer 2s

Bitcoin has Layer 2s like: - Lightning Network (for fast, cheap payments)

Ethereum has Layer 2s like: - Arbitrum - Optimism - Polygon - Base - And many more...

Each one is essentially its own CDN that periodically syncs with the main chain for security.

This is exactly how human societies already work.

15.4.2 The United States

Federal Government (Layer 1): - Makes big decisions (constitutional amendments, war, trade deals). - Slow (Congress takes forever). - Expensive (huge bureaucracy). - Final arbiter (Supreme Court).

State Governments (Layer 2): - Make local decisions (traffic laws, local taxes, education). - Fast (state legislatures move quicker). - Cheap (smaller bureaucracy). - Defer to federal for disputes (federal law overrides state law).

Both work together. States handle day-to-day governance. Federal government handles major coordination.

I will not repeat myself with the EU example, it’s similar.

But CDNs have new features never seen in history. Imagine every week each State updates their data to the federal CDN, and so, as that one is decentralized, it remains there forever.

15.4.3 A Corruption Example

Imagine inside a State, because it is a more centralized CDN and easier to corrupt, there is a case of corruption where money is stolen or misused. Then you could use last week’s data as a fresh starting point, a consensual moment where everyone agrees on the state of things, and from there you can start investigating backwards to see how deep and for how long the corruption actually happened.

In normal traditional cases, the files that prove suspicious movements through weeks and weeks might be “lost” or destroyed by those in power who got corrupted. But with blockchain checkpoints, you only have one week to cover your tracks. You have to hurry, and in a serious legal system, if you try big crimes quickly, chances are you get caught.

15.4.4 Another Example

You loan money to Bob, but Bob is friends with Alice, a very rich person who also happens to be a politician and is well connected to the people who run this quick but centralized CDN. Let's say Bob cannot pay you back. He's in trouble, but calls Alice: "Hey, can we just 'lose' the records of this loan and pretend it never happened? Can we abuse the consensus algorithm to even re-write history a bit?" A bit, notice the pun.

Alice agrees and uses her influence to get the centralized CDN to "lose" the records of this loan. You are out of luck.

But, in a system with periodic updates to L1, we can see that Bob actually took a loan from you 2 weeks ago and now it has vanished. Investigation time for the federal authorities.

The decentralized Layer 1 acts as an immutable checkpoint. Even if Layer 2 is compromised, Layer 1 preserves the truth.

15.4.5 Another Another Example

Imagine you borrowed from someone, and that lender knew you could not pay the loan back but still gave you the loan anyway—something similar to the 2008 bubble. Well, imagine that happening at a State level in the US, and again, the centralized system tries to excuse itself with, "well... we did not know, how would we have known?" Well... the data is public. And look at the Federal CDN: it is clearly visible that you were taking very risky, clearly irrational decisions that led lots of investors and people to bankruptcy.

Oh wait, everything is public... Is that good? Some people care about privacy.

15.5 The Temporary Bad News: Trade-Offs

Layer 2s aren't perfect. They introduce complexity.

15.5.1 Technical Complexity Is Real

Technically, L2s can use very diverse algorithms to run, and we still need to find a general way to easily connect them to one another. People are working on this, but for now, technical complexity is a real thing making the scenarios described challenging to execute.

These systems do exist, but because they are complex, it is hard for normal people to understand them and therefore use them for anything meaningful.

This creates a barrier to adoption. The technology works, but the user experience is still being figured out.

Imagine that to connect to the internet you had to do it completely differently for each brand of computer you use. That would certainly slow things down.

15.5.2 Cognitive Load: Too Many Choices

Beyond technical complexity, there's also the cognitive load it creates.

Users have to choose which Layer 2 to use. Do I use Arbitrum? Optimism? Polygon? Base? It's like choosing which state to live in—not everyone understands the exact differences and it takes time to do so.

Each Layer 2 has different trust assumptions (some are more centralized than others), different speeds and costs, different security models, and different applications built on them. **How is a normal person supposed to choose wisely?**

Right now, most people don't. They stick to Layer 1 or use whatever their favorite app recommends. This defeats some of the purpose.

If you want to independently move your money around all these CDNs, you have to really know what you are doing. What is the chainID? Is this node provider reliable? Do I need to create a new wallet for this chain? Is the token or coin I want to use available in this chain? Is the code in this chain as secure and reviewed by security experts as in this other chain? Where can I find a simple user interface to move funds between the chains with one or two clicks? Is this website I found secure? Who runs it? Nice, I found another chain—let's gather all this information again...

It's like moving residency—possible, but not instant or free.

No one has time for this, just nerds like the author. As said, technical advancements like creating new consensus rules are going to simplify the process in the future, and users will not have to know that much about all the details at all. But that's just under construction.

15.5.3 Different Trust Assumptions And Software Accessibility

Some Layer 2s are more centralized than Layer 1. You're trusting the Layer 2 operator to some or to a full degree.

It's a trade-off: more speed and lower cost, but a slightly or completely different security model.

What if the operator just disappears and you did not have time to move your money back to the safe CDN? What if the developer who made the website disappears and now, even if it is technically possible, you don't know how to move your funds back?

Not everyone in this world is a software engineer, and even fewer people understand and can quickly interact with CDNs. And it is not realistic to expect everyone to become one for the sake of a more effective financial system.

What is the solution then? As said, technicians are working on it—on websites anyone can run on their machines even without internet, on apps that are open-source with code that is public and everyone can verify, that they can download and use forever on their phone to move funds around safely even if their favorite website is down.

There have even been invented cryptographic mechanisms where, even if the L2 breaks, people can just move their funds to the main CDN without having to interact with the broken one.

It is unrealistic to expect everyone to become a software engineer, but it is not unrealistic to create open-source software usable and accessible by anyone at global scales. Linux, a free operating system that works perfectly, is free and for everyone, anywhere. And so on and so forth—there is lots of software capable of this.

It is true that these softwares require a small effort; they are not just available in the Play Store where you install them in one click. But this extra thing or two you have to learn is minimal and

anyone can do it in one or two afternoons, especially the new generations who were born around digital tools. And it is likely that someone creates a way to even delete this small barrier to entry in the future.

Therefore, give us time and we will give you collective freedom. Or put a small effort in understanding how to download code from places like GitHub. Modern AIs can explain how to do this pretty well.

If you are a software engineer reading this, I encourage you to use your knowledge and build for the CDN niche. Thanks so much, you are very welcome, we will never have enough people working—it just creates more decentralization and better systems. And please, do not forget the purpose, do not forget that decentralization is what keeps the party going.

15.5.4 Industry Age Context

This industry is just about **16 years old** as of **2025** if you count since Bitcoin was created—very young—and lots of code is still to be written, and to write it securely slows down the process. Mess something up in this industry and you get some irrecoverable hacks, as we have explained in earlier chapters.

15.5.5 Fragmentation

If everyone uses different Layer 2s, the network effect is split.

It's like how the EU benefits from all nations coordinating, but each nation speaking a different language creates friction.

If your money is on Arbitrum and your friend's money is on Optimism, transacting between you two is harder than if you were both on the same network.

It is like sending a package to another State instead of sending it to someone in the closest city.

15.6 Summing up

Local -> Regional -> National -> Global coordination.

- You coordinate with your family (very fast, very local).
- Your city coordinates internally (fast, local).
- Your state coordinates with other states (slower, regional).
- Your nation coordinates with other nations (slow, global).

Layer 2s are the same pattern applied to CDNs.

We're not, at social relationship levels, inventing something new—we're recognizing that coordination naturally happens at multiple scales.

Most of your daily life doesn't need global coordination. Only some things do.

15.7 Security Is Shared Globally

Imagine that in traditional systems, every time you had a dispute, the best judge, lawyers, and detectives in the world helped you. That is impossible.

In a way, not anymore. As we saw, we can periodically update the data to the secure system, creating unerasable history, evidence—readable and analyzable at the speed of electricity, anywhere in the world.

Societies within societies, all working together and for each other.

Ethereum doesn't need to process every transaction. It just needs to be the final arbiter when disputes arise or when checkpoints are needed.

Bitcoin doesn't need to record every coffee purchase. It just needs to settle final balances when Layer 2 channels close.

This is how coordination scales.

Not through forcing everyone to agree on everything, but through allowing local groups to coordinate quickly while maintaining a global layer for truth and security.

This creates the idea of Ethereum as the global trust machine. Why do you live relaxed even if literally anyone can hurt you when you go outside? Because we have designed our system to have “automated” mechanisms, like police and judges, to help you when that happens. These traditional systems are also trust machines that build trust. And when you trust you can relax, and then the better the sex—same applies in finance and data coordination. Furthermore, these systems can also be influenced and improved by you in case they get something wrong, via voting for new laws, etc.

In an inverse phrasing, they also generate the lack of needing to trust, because you know if someone misbehaves, they will be punished. This is what the industry calls “trustless.”

By the way, the trustless manifesto and d/acc philosophy from Vitalik Buterin reflect very well how the main forces guiding Ethereum have these intentions I've been describing. I encourage you to read them.

As you can see, CDNs are very similar: valid consensual agreements on what is the official information, in which everyone can participate and even propose changes if things seem not to work. And unlike traditional systems, these systems can be global and actually automated.

This is the current state of thought of the industry and a glimpse into the possibilities it opens. I encourage you to keep learning about it—as you can see, it is technological advancement that grants us new capabilities we just did not have before.

But now, wait, have you noticed? We all are naked.

15.8 The Last Downside, Privacy

Houston, there's still a problem: **All these transactions and data are still public.**

Everyone can see your balance. Everyone can see who you transact with. Everyone can trace your financial history.

Criminals can see how much money you have and decide whether to target you.

Corrupted governments can too. I mean, they are a subset of the criminals example above.

Even your neighbors can see your spending habits, can envy you, and create social pressure.

Insurance companies can see your data and decide to charge you more or less for no real reason related to the actual insurance policy.

Employers can see what you spend on before hiring you, discriminating based on your personal life.

Authoritarian regimes can track dissidents' donations to opposition groups.

Stalkers and abusive ex-partners can monitor your financial movements and location patterns.

Companies can track your purchases and build invasive advertising profiles without your consent.

Good for auditability and responsibility generation. Bad for privacy and protection.

Is there a way to have both privacy and verification?

Can you prove you have enough money to make a purchase without revealing how much you have?

Can you prove you're old enough to enter a bar without showing your exact birthdate?

Can you prove you voted without revealing who you voted for?

This seemed mathematically impossible for a long time. If you want to prove something is correct, don't you have to show it?

Turns out, no. And that's the topic of our next chapter.

Key Insight: The blockchain trilemma: you can only have 2 of 3 (decentralization, security, scalability). Bitcoin and Ethereum chose decentralization + security, sacrificing speed (~7-15 tx/sec vs Visa's ~24,000). But we don't need global consensus for everything—most coordination can be local. Layer 2s are smaller CDNs on top of Layer 1: Layer 1 is slow, secure, global (federal government), while Layer 2s are fast, cheap, local (state governments). Examples: Lightning Network (Bitcoin), Arbitrum/Optimism/Polygon/Base (Ethereum). They periodically checkpoint to Layer 1, creating an immutable record that prevents corruption from being hidden. Trade-offs: technical complexity (hard to connect L2s), cognitive load (too many choices), different trust assumptions, coordination overhead, fragmentation. This mirrors human society: local -> regional -> national -> global. Consensus can be nested—societies within societies. Most daily life doesn't need global consensus, only checkpoints for truth and security. But all transactions are still public—can we have privacy AND verification?

Next, we'll explore Zero-Knowledge proofs—proving you know something without revealing what you know. This seemed impossible until the 1980s, and impractical until the 2010s, but it's now the missing piece for private, verifiable coordination.

16

Chapter 16: Zero-Knowledge Proofs - Proving Without Revealing

Can you prove you know something without showing anything about what you know?

In Chapter 15, we identified a critical problem: **all transactions on blockchains are public.** Everyone can see your balance, everyone can trace your history, and criminals, governments, neighbors, and employers alike can monitor your financial life.

So here's the question: Can you prove something is true without revealing what that something is?

Can you prove you have enough money to buy a house without revealing your total wealth? Can you prove you're over 21 without showing your exact birthdate? Can you prove you know a password without typing it?

For most of human history, the answer seemed obvious: **No. If you want to prove something, you have to show at least some information about it.**

It really seems impossible, feels like magic, even illogical. But it's just math. Think about it for another second: how can I show you I'm over 18 while revealing nothing else? How can I say "Hey, I am over 18" and then, without you needing to believe me, you simply KNOW I am? It's crazy, and therefore, fascinating.

In the 1980s (in **1985** specifically), mathematicians discovered something shocking: **You can prove you know something without revealing what you know.**

This is called a **Zero-Knowledge Proof (ZKP)**. The name pretty much describes exactly what it does: proving stuff while revealing zero knowledge about it.

16.1 What Is a Zero-Knowledge Proof?

A Zero-Knowledge Proof is a way for one person (the prover) to convince another person (the verifier) that a statement is true, without revealing any information beyond the truth of that statement.

Example:

- **Statement:** "I know the password to this account."
- **Traditional proof:** Type the password -> Verifier sees the password.

- **Zero-Knowledge proof:** Run a mathematical protocol -> Verifier is convinced you know the password, but learns nothing about what the password is.

The verifier learns **ONLY** that the statement “I know the password” is true. Nothing else.

16.2 A Simple And Classic Analogy: The Colorblind Friend

Imagine you have a friend who is colorblind. You have two balls: one red, one green. They look identical to your friend.

You want to prove to your friend that the balls are different colors, but **without revealing which is red and which is green.**

Here's how:

1. Your friend holds both balls behind their back and randomly swaps them (or doesn't).
2. They show you the balls again and ask: “Did I swap them?”
3. If the balls are truly different colors, and you are not colorblind, you'll always answer correctly.
4. If the balls were the same color, you'd only guess correctly 50% of the time.

Repeat this 20 times.

If you answer correctly every time, your colorblind friend becomes convinced: “The probability of you guessing correctly 20 times in a row by chance is 1 in 1,048,576. You must actually see a difference in the colors.”

You've proven the balls are different colors without ever saying which one is red or green.

Your friend learns: “The balls are different.”

Your friend does NOT learn which one is red or which one is green—nothing beyond the truth of the statement itself.

This is the essence of Zero-Knowledge Proofs.

16.3 The Mathematical Breakthrough

In 1985, three researchers (Shafi Goldwasser, Silvio Micali, and Charles Rackoff) published a paper proving that Zero-Knowledge Proofs are mathematically possible.

This was shocking. Most people assumed: “To prove something, you must reveal it.” But math said otherwise.

Remember how we got digital identities in Chapter 6? We used **one-way functions**—mathematical operations that are easy to compute in one direction but nearly impossible to reverse:

- Easy: $\text{hash}(\text{"password123"}) = \text{d3f8e9...}$
- Hard: $\text{d3f8e9...} = \text{hash}(\text{"????"})$ (reverse it)

Zero-Knowledge Proofs use even more advanced mathematics. The math behind ZKPs is taught in advanced university courses (graduate-level cryptography), PhD programs in mathematics and computer science, and specialized research labs.

It involves: - Elliptic curve cryptography. - Polynomial mathematics. - Group theory and abstract algebra. - Probabilistic algorithms. - Advanced number theory.

If you're curious and want to understand the full technical details, here's what you'd need to study: - Modular arithmetic. - Finite fields. - Discrete logarithms. - Pairing-based cryptography. - Polynomial commitments. - Fiat-Shamir heuristic. - zk-SNARKs and zk-STARKs (specific ZKP constructions).

But honestly, this would require another entire book, or books.

The math is extremely complex—even most computer scientists don't fully understand all the details. But again, the more people who understand this technology, the more decentralized and stronger the system becomes, so I encourage you to learn all of the above if you feel like it. It will take you some years if you have no background.

What matters for this book is understanding what Zero-Knowledge Proofs can do, not really how they do it.

So let's focus on the properties.

16.4 The Three Properties of Zero-Knowledge Proofs

For something to be a true Zero-Knowledge Proof, it must have three properties:

16.4.1 1. Completeness

If the statement is true, an honest prover can convince an honest verifier.

- If you actually know the password, you can prove it.
- If you actually have \$100,000 in your account, you can prove it.
- If you're actually over 21, you can prove it.

A correct proof always works.

16.4.2 2. Soundness

If the statement is false, no cheating prover can convince the verifier (except with negligible probability, like 0.000000000000000000000000001%—not in the lifetime of a universe or several of them).

- If you DON'T know the password, you can't fake the proof (except by getting extremely lucky).
- If you DON'T have \$100,000, you can't trick the verifier into thinking you do.

You can't lie with a Zero-Knowledge Proof. Or at least, lying is so astronomically unlikely that it's effectively impossible.

16.4.3 3. Zero-Knowledge

The verifier learns nothing except that the statement is true.

- They don't learn the password.
- They don't learn your exact balance (just that it's $\geq \$100,000$).

- They don't learn your birthdate (just that you're ≥ 21 years old).

No information leaks beyond the truth of the statement itself.

16.5 How Does This Help Blockchains?

Blockchains have a transparency problem.

Zero-Knowledge Proofs offer a solution:

Instead of sending to the network: "Alice sent 5 BTC to Bob," you broadcast:

"A valid transaction occurred. Here's a Zero-Knowledge Proof that: - The sender had enough balance. - The amounts are correct. - No coins were created out of thin air. - The transaction follows all the rules."

The network can verify the transaction is valid without seeing: - Who sent it. - Who received it. - How much was sent.

Privacy + Verification. Both at once.

You can even have selective privacy: you might not care about hiding the sender, but want to hide the amount. Or vice versa.

16.6 Real-World Examples

16.6.1 Example 1: Private Transactions

Zcash is a cryptocurrency that uses Zero-Knowledge Proofs to enable private transactions.

- You can send money to someone.
- The network verifies the transaction is valid.
- But no one (except you and the recipient) knows how much was sent or who was involved.

Public ledger. Private details.

16.6.2 Example 2: Proving You're Over 18

You want to enter a bar. The bouncer needs to verify you're over 18.

Traditional method: - Show your driver's license. - Bouncer sees: your name, address, birthdate, photo, license number, organ donor status, etc.

With Zero-Knowledge Proofs: - Your phone generates a ZKP that proves: "This person is ≥ 18 years old." - Bouncer scans it and is convinced. - Bouncer learns: "This person is over 18." - Bouncer does NOT learn: your exact age, name, address, or any other detail.

Minimal information disclosure.

16.6.3 Example 3: Private Voting

You want to vote in an election. The system needs to verify: - You're eligible to vote. - You haven't voted already. - Your vote is recorded correctly.

Traditional digital voting has problems: - If votes are public, no privacy. - If votes are secret, how do you verify they were counted correctly?

With Zero-Knowledge Proofs: - You generate a proof: "I am an eligible voter, and I cast a valid vote." - The network verifies the proof. - Your vote is counted. - No one knows who you voted for, but everyone can verify the total count is correct.

Privacy + Auditability.

16.6.4 Example 4: Proving Solvency Without Revealing Balances

A cryptocurrency exchange claims: "We have enough funds to cover all user deposits."

Users want proof, but the exchange doesn't want to reveal: - Exactly how much they hold. - Their wallet addresses (security risk). - Individual user balances.

With Zero-Knowledge Proofs: - The exchange generates a proof: "Total user deposits = X. Total exchange holdings \geq X." - Users can verify the proof. - Users learn: "The exchange is solvent." - Users do NOT learn: exact balances, wallet addresses, or other sensitive details.

Transparency without exposure.

16.7 Why This Matters

This is revolutionary. For most of human history, you had to choose:

Either: - **Transparency:** Everyone can verify everything, but there's no privacy.

Or: - **Privacy:** You keep secrets, but no one can verify your claims.

You couldn't have both.

Banks are private (you can't see others' balances) but not transparent (you can't audit the bank's reserves—and if you do manage to audit them, it's a long, slow process that's vulnerable to tampering).

Blockchains are transparent (you can verify everything) but not private (everyone sees your balance).

Zero-Knowledge Proofs break this trade-off.

You CAN have both privacy and verification. This opens entirely new possibilities: - Private financial systems that are still auditable. - Anonymous credentials that are still verifiable. - Secret votes that are still provably counted correctly. - Confidential medical records that can still prove you're vaccinated.

Privacy and trust, together.

In practice, a Zero-Knowledge Proof represents a computation, a program. So, in theory, if anything can be computed, it can be proven in zero-knowledge.

16.8 The Catch: Complexity and Performance

Zero-Knowledge Proofs are **incredibly complex** and **computationally expensive**.

16.8.1 Complexity

The math is so advanced that most developers don't fully understand it yet, implementing ZKPs correctly is extremely difficult, and bugs in ZKP systems can be catastrophic (breaking either privacy or security).

Very few people in the world can build these systems correctly.

This creates a barrier. Unlike basic cryptography (which is now well-understood and standardized), ZKPs are still cutting-edge research. They only started becoming practically useful in the 2010s, and are still evolving rapidly.

16.8.2 Performance

Generating Zero-Knowledge Proofs quickly requires significant computational power.

But they're improving rapidly.

New ZKP systems (zk-SNARKs, zk-STARKs, Plonky2, and more) are making proofs smaller, faster, and easier to generate.

What took minutes in 2015 now takes seconds in 2025 in many cases. What takes seconds today might take milliseconds in 2030.

Performance is improving exponentially.

16.9 The Future: Private Coordination at Scale

Imagine a world where:

- You can prove your income to a lender without revealing your exact salary.
- You can prove your medical history to a doctor without exposing sensitive details.
- You can vote in elections where the results are publicly verifiable, but your vote is private.
- You can transact on a public blockchain without revealing your balance or transaction history.
- Governments can prove they're following the law without revealing state secrets.
- Companies can prove they're not doing illegal stuff without revealing trade secrets.

Zero-Knowledge Proofs make all of this possible.

They let us build systems that are:

- **Verifiable:** You can prove claims are true.
- **Private:** You don't reveal any unnecessary information.

Mix it with CDNs for:

- **Trustless:** No central authority needs to be trusted.

This is a fundamentally new capability.

Before ZKPs, you always had to choose: transparency or privacy. Now you can have both.

Combined with blockchains (public, verifiable, tamper-proof ledgers), Zero-Knowledge Proofs enable **private, verifiable coordination at global scale**.

16.10 But We're Not There Yet

Zero-Knowledge Proofs are still:

- **Complex:** Hard to build correctly.
- **Slow:** Computationally expensive.
- **New:** Not yet widely understood or adopted.

Most CDN systems today do NOT use ZKPs. Bitcoin doesn't. Ethereum's main chain doesn't (though Layer 2s are starting to). However, Ethereum is actively exploring ZKPs for scalability and privacy, recently in early 2026, Vitalik, the inventor of Ethereum said they solved the blockchain trilemma using ZKPs and now it is just a matter of writing the code in a very safe way.

Why? Because they're hard to implement, slower than regular transactions, and the technology is still maturing.

But the progress is rapid. What seemed impossibly slow in 2015 is practical in 2025.

ZKPs are one of the most important innovations in cryptography in the last 40 years.

And they're just getting started.

16.11 A Note on Quantum Computers

Remember in Chapter 8 we talked about quantum computers potentially breaking certain types of cryptography?

The same applies to Zero-Knowledge Proofs.

Some current ZKP systems could be vulnerable to quantum computers: - zk-SNARKs based on elliptic curve pairings could be broken by quantum computers using Shor's algorithm. - These rely on mathematical problems (like discrete logarithms) that quantum computers can solve efficiently.

But some ZKP systems are believed to be quantum-resistant: - zk-STARKs use hash functions and don't rely on elliptic curves or pairings. - Hash-based ZKPs should remain secure even against quantum computers. - Lattice-based cryptography (another approach) is also being explored for post-quantum ZKPs.

The good news: The pattern repeats. Researchers are actively working on quantum-resistant Zero-Knowledge Proofs. By the time quantum computers become a real threat, we'll likely have transitioned to quantum-safe ZKP systems.

Key Insight: Zero-Knowledge Proofs let you prove a statement is true without revealing any information beyond the truth of the statement itself. Three properties: completeness (true statements can be proven), soundness (false statements can't be faked), zero-knowledge (no extra information leaks). This breaks the historical trade-off between transparency and privacy. You can now have both: prove you have enough money without revealing your balance, prove you're over 21 without showing your birthdate, prove a transaction is valid without revealing who sent it or how much. The math is extremely advanced (graduate-level cryptography, polynomials, elliptic curves, abstract algebra), and the technology is computationally expensive, but improving rapidly. Combined with blockchains, ZKPs enable private, verifiable coordination at global scale. This is one of the most important cryptographic breakthroughs of the last 40 years, and it's just getting started.

Next, we'll step back and look at the bigger picture: what do all these technologies mean for society, coordination, and human freedom? How do CDNs, smart contracts, Layer 2s, and Zero-Knowledge Proofs change the landscape of power, trust, and collective decision-making?

Chapter 17: What This All Means for Humanity

This technology makes physically impossible things possible.

We have covered a lot of ground together: bits, algorithms, cryptography, consensus mechanisms, blockchains, smart contracts, Layer 2 solutions, and Zero-Knowledge Proofs.

But let's zoom out for a moment.

What have we actually built here?

This isn't just "digital money." It isn't just "smart contracts." It isn't just another tech trend.

This is coordination technology for strangers at the internet scale.

And it makes things possible that were literally, physically impossible before **2009**, when the Bitcoin network launched.

This is the beginning of a new chapter in human history that allows us to coordinate and alter the traditional behavior in some of our societal structures in ways that once sounded impossible and illogical.

17.1 What Was Impossible Before

Throughout human history, certain things required trust in centralized intermediaries.

Want to send money across borders? You needed banks to verify balances and process transfers. They could freeze your account, reverse transactions, charge fees, or deny service entirely. You had no choice but to trust them.

Want to prove ownership of a digital asset? You needed a central registry—a game company, title office, or stock exchange. They controlled your assets, and if they shut down, your ownership disappeared. At best, it might come back after long, expensive delays and legal battles. Digital ownership didn't really exist—only permission to use.

Want to coordinate with strangers without a mediator? Impossible. Someone had to be the trusted referee. Contracts required courts to enforce, and agreements required intermediaries to verify. A human in the middle was mandatory.

Want to create money without government? Governments monopolize currency creation. Private currencies either got shut down or required trust in the issuer. Money was a matter of privilege classes or State control.

Want to prove something without revealing it? Until Zero-Knowledge Proofs, this was mathematically impossible. Privacy required hiding while verification required revealing—you couldn’t have both.

Want to create a voting system that can’t be tampered with? Voting required trusted authorities to count and verify. Paper ballots could be lost, altered, or miscounted. A fully, mathematically proven and mathematically accurate, trustworthy voting system was unattainable.

Want to make sure the winners do not re-write history? Centralized databases could be altered or deleted by those in control. Records could be changed, erased, or hidden. An unchangeable, auditable history was out of reach.

Want to gather money for a new business idea from lots of investors around the world? You needed intermediaries—banks, brokers, legal teams. Regulations made it expensive and slow. Permissionless crowdfunding on a global scale was unachievable.

All of these things were physically impossible.

And then, starting in 2009 with Bitcoin, they slowly became possible.

Not easy. Not perfect. But **possible**. That is already a leap of infinite size.

17.2 What Changed?

Mathematics + Engineering + Cryptography + Incentives + Social Consensus = Trustless Coordination

Let’s break that down:

17.2.1 Mathematics

Cryptographic functions—hashing, signatures, Zero-Knowledge Proofs—provide **verifiable truth without requiring trust**.

You don’t trust me that the signature is valid; you verify it mathematically. You don’t trust me that the hash is correct; you compute it yourself. You don’t trust me that I’m over 18; you verify the Zero-Knowledge Proof.

Math doesn’t lie. Math doesn’t have opinions. Math is the same everywhere.

17.2.2 Incentives

Game theory aligns participants’ interests.

Miners and validators profit from following the rules and lose from breaking them. Attackers must spend enormous resources to succeed, and even if they do, they destroy the very value they’re attacking—only externally-funded threats make sense.

The system assumes selfishness and mathematically turns it into collective benefit.

17.2.3 Social Consensus

At the end of the day, **humans decide which rules to follow.**

The code doesn't enforce itself—people choose to run it. Value doesn't come from the technology—it comes from people agreeing it has value. The consensus algorithm doesn't force anyone—it coordinates voluntary participants.

Technology enables. Humans decide.

17.3 The Power Dynamics Shift

This creates a fundamental shift in how power coordinates.

17.3.1 Before: Centralized Gatekeepers

Banks control your money. They can freeze accounts, reverse transactions, and deny service.

Governments monopolize currency. They can inflate supply, confiscate wealth, and control access without much friction.

Platforms own your data. They can censor content, ban users, and change terms arbitrarily.

Corporations mediate agreements. They can charge rent, change rules, and shut down services.

You had to trust them. There was no alternative.

But let's not be egoistic—these organizations also had to trust each other. It is no surprise how politicians often attack each other; they do not necessarily trust each other either. It is no surprise that brokers on Wall Street often try to scam each other or take advantage of each other's information to profit.

This is not a revolution from the people to the ruling class—it goes further than that. It is a revolution of trust-management itself that we can all benefit from.

But as with all technological advancement, it will be useless if we do not learn some basics of usage and understanding of the implications. Like when cars were invented, we needed to learn to drive and agree on similar traffic rules that apply worldwide. We created a revolution in how we moved.

This is the revolution in how we trust.

A revolution in one of the deepest aspects that dictate how we, as a species, coordinate.

17.3.2 After: Distributed Coordination

CDNs (Blockchains): No single controller. Voluntary participation. Transparent rules. Math enforces agreements.

Smart contracts: Code executes automatically. No intermediary needed. No one can stop it once deployed.

Self-custody: You control your keys, you control your assets. No one can freeze or confiscate without your private key.

Voluntary global participation: Don't like the rules? Fork. Exit. Join another network. No one can force you to stay.

You don't need to trust anyone. You verify the math. You choose which network to participate in.

This doesn't eliminate power nor certain phenomena emergent from human organization at scales, like the formation of oligarchies. But it does change who can access power and how it is negotiated.

17.3.3 The Anti-Gaslight Machine

Remember the anti-gaslight machine from Chapter 12?

Gaslighting is when someone makes you doubt your own memory or perception of reality by lying repeatedly until a "truth" becomes accepted.

Governments do this. Corporations do this. Abusers do this.

They delete records. They change documents. They deny things happened. And if you can't prove it, you are powerless. Even if you can prove it, in the best case scenario, you still have to "pray" the courts are not bribed—that vital evidence isn't somehow "lost" or "accidentally destroyed."

And sure, all this assuming they are not putting an entire system of glowing machines with short and emotionally charged texts and videos in order to distract your thoughts and, effectively, control them.

Anyway, blockchains make all these things easier to prove.

If a government tries to rewrite history, everyone's copy of the blockchain still shows the truth.

If a majority forks the network (like Ethereum did with the DAO), the minority can keep the original chain (Ethereum Classic). The disagreement is visible forever.

If a corporation claims "we never did that," the blockchain says: "Block 1,920,000 proves you did."

No one can hide tyranny. No one can erase the past. At least the economic history.

This is a fundamentally new property of human coordination systems.

17.4 Real-World Implications

Let's get concrete. What does this actually enable?

17.4.1 Money: Cannot Be Frozen, Censored, or Arbitrarily Inflated

Bitcoin has no CEO. No one can shut it down, freeze your account, or reverse your transactions.

As long as you control your private key, you control your money.

Governments can't arbitrarily inflate Bitcoin's supply. The 21 million cap is enforced by code and social consensus, not by political decisions.

This doesn't mean Bitcoin is perfect money. It's volatile, slow, and hard to use for everyday transactions.

But it's **uncensorable, unseizable, and uninflatable**. That was impossible before.

17.4.2 Governance: Transparent Voting, Credible Neutrality, DAOs

Decentralized Autonomous Organizations (DAOs) use smart contracts to coordinate groups without traditional hierarchy.

Members vote on proposals. Code executes decisions automatically. Transparent rules are enforced by math and code.

Is this perfect? No. DAOs have governance problems—voter apathy, plutocracy, coordination failures. Nothing new under the sun here; the iron law of oligarchy still applies.

But they enable **permissionless participation in governance**. Anyone can join. Anyone can propose. No one can be excluded arbitrarily.

17.4.3 Identity: Own Your Data, Portable Across Platforms

Right now, your identity is fragmented. Facebook owns your social graph. Google owns your email. Banks know your financial history. Each platform owns your data.

With CDNs, you can own your identity.

Your credentials, reputation, and data live on-chain or in Zero-Knowledge Proofs. You prove what you need to prove without revealing everything. You take your identity with you across platforms.

Is this ready today? Not quite for normal people usage. But the infrastructure is being built.

17.4.4 Coordination: Layer 2s = Mini-Societies

As we learned in Chapter 15, Layer 2s enable **nested coordination**—societies within societies.

Different communities can have different rules while still coordinating with the global layer when needed.

This is how human society already works (cities, states, nations, international agreements). Now we can do it with databases.

17.4.5 Privacy: Zero-Knowledge = Prove Without Revealing

As we learned in Chapter 16, Zero-Knowledge Proofs enable **privacy with verification**.

You can prove you're eligible to vote without revealing who you are. You can prove you have enough money without revealing your balance. You can prove you're qualified for a loan without revealing your salary.

Dignity + Freedom.

You don't have to choose between privacy and participation.

17.5 Flami's Farewell: Leaving the Propaganda Zone

Hellow, I'm Flami ! We are slowly leaving the propaganda zone. I hope you enjoyed the ride. Bumps on the road ahead, please fasten your intellectual seatbelt.

This technology is powerful. It does enable things that were literally impossible before, and therefore it represents technological and even intellectual progress.

But what you have been reading so far in this chapter has been glossed over. This has been the reflection of the wet dreams of the “crypto” industry (CDNs industry).

There are many caveats. Many. For marketing purposes, they are omitted most of the time, but they are real.

It makes sense that when introducing people to new complex technologies that are revolutionary, you need to simplify—sometimes falling into the propaganda realm.

But that does not help in the medium and long term.

The next chapter will give you the full reality check—all the nuances, limitations, and uncomfortable truths about this technology. The trade-offs. The failures. The exaggerations. The honest assessment of what works, what doesn’t, and what’s still uncertain.

Before we get there, let me leave you with the philosophical core:

17.6 The Philosophical Core

Let’s distill this to first principles.

17.6.1 Consensus Is Social, Not Technical

The technology doesn’t enforce itself.

Bitcoin’s code says “21 million coin limit.” But if everyone agreed to change it, they could.

The consensus algorithm doesn’t force anyone. It coordinates willing participants.

Code is law, but only because humans agree to run the code.

17.6.2 Value Is Consensual In Complex Worlds

Gold has value because people agree it does. The US Dollar has value because people agree it does. Bitcoin has value because people agree it does.

Value in big complex societies predominantly becomes social consensus. It always has been.

Some things have value for their usage, like food, but when societies grow and we no longer crave survival, value mainly becomes a social construct for coordination.

Bitcoin just makes this transparent. The value isn’t hidden in allegedly completely independent central bank decisions or government decrees. It’s visible in the market, in the nodes, in the forks.

17.6.3 Coordination Is Voluntary

No one can force you to participate in a CDN.

You choose which network to join. You choose which software to run. You choose when to exit.

This is fundamentally different from traditional systems: - You can't opt out of your government's currency (try paying taxes in Bitcoin). - You can't opt out of bank regulations (try opening a bank without a license). - You can't opt out of platform rules (try negotiating with Facebook's terms of service).

With CDNs, exit is always an option. Fork. Join another network. Start your own.

This doesn't mean there are no power dynamics. Network effects matter. Knowledgeable people have advantages. But the **ability to exit and the ability to truly own change the game.**

17.6.4 Technology Enables, Humans Decide

CDNs don't have a political opinion.

It's a tool. Like fire, electricity, or the internet.

You can use Bitcoin to escape authoritarian capital controls. You can use Bitcoin to evade taxes and fund crime. You can use Bitcoin as speculative investment. You can use Bitcoin as a philosophical statement about money.

In the same way you can use the State to create good public services that avoid famine or use it to steal public funds and wage unnecessary wars. The systems are tools, the people who use them are the ones who add the ethical factor.

The people who run Bitcoin do have political opinions. Like choosing that a limited supply is the good thing to do.

The technology is neutral. Humans give it meaning.

Not because there are corrupted governments or States that are "evil" dictatorships out there we must never use the idea of a State in other contexts. The same applies for Bitcoin and CDNs, just because some people use those systems for illegal things it does not mean we should reject them entirely.

17.7 The Invitation

This isn't a story about "them"—the developers, the miners, the institutions.

This is about us. All of us.

Because CDNs only work if people choose to participate. They only have value if people agree they have value. They only change the world if people use them to change the world.

You can participate however you want:

- **Build:** Write code. Create applications. Improve the infrastructure.
- **Critique:** Point out flaws. Identify risks. Push for better solutions. Call out scams and hype.
- **Regulate:** Work in government to create sensible policy.
- **Use:** Send transactions. Participate in DAOs. Experiment with the technology.
- **Educate:** Teach others. Write. Explain. Spread understanding.

Just do it with understanding, not hype.

Ask the critical questions: - Does this problem need a CDN, or is there a simpler solution? - Does the cost of a more complex solution outweigh the benefits? - Am I being sold hype, or is there real value here? - What are the trade-offs? What are the risks? - Who benefits? Who loses? - Who are the users? Am I building for real people with real needs?

Be skeptical. Be curious. Be thoughtful.

17.8 The Core Message

This is REAL technological advancement.

Not hype. Not magic. Not a get-rich-quick scheme.

This technology allows humans to do things that were literally, physically impossible before:

- Coordinate trust in such a new way that we even need a new term: “trustless”.
- Own without intermediaries.
- Prove without revealing.
- Create money without governments.
- Record history without central authority.

These are new capabilities for humanity.

We didn't have them in 2008. We have them now.

But with great power comes great responsibility.

This technology can be used for good or evil. It can liberate or enable crime. It can decentralize power or concentrate it in new ways.

It's up to us—not developers, not governments, not corporations—all of us.

What we build with these new capabilities will define whether this technology becomes a net positive or just another tool for the powerful. Just like with any other technology.

17.9 We've Unlocked New Forms of Coordination

For most of human history, lots of processes requiring coordination also required hierarchy **Because someone had to be the trusted referee.**

CDNs offer an alternative: **Coordination through math, cryptography and incentives that define a social consensus.**

Not perfect. Not always better. But **possible**. The hierarchy tree is flattened to its minimum in a way that is effectively “hierarchyless” in practice, as no rational action inside the organization would abuse their privileges—making everyone effectively equal within the group, just with different jobs.

With nuances, it is like a policeman in traditional organizations. Policemen do have privileges, but if they break the law, being a policeman won't protect them from being prosecuted. The nuance is that a policeman breaking the law sometimes does not make the whole State collapse the next day; in a CDN, it would.

And now that it's possible, we have choices we didn't have before.

What we choose to build is up to humanity.

Now I ask you: will you build anything? How will you use it?

What will you choose?

Again, enough of the bright potential futures—they are just potential for now. Let's see the raw reality in the next chapter before concluding this book.

Key Insight: This technology enables new forms of coordination that were literally impossible before 2009. It shifts power from centralized gatekeepers to distributed networks. It makes history auditable, assets ownable, privacy verifiable, and money uncensorable. The technology is neutral—humans decide how to use it. This depends on all of us: to build, critique, regulate, use, or ignore it wisely. We've unlocked new capabilities. What we build with them will define whether this becomes a net positive for humanity. But before we celebrate, we need a reality check—and that's what comes next.

Next: Chapter 18 will give you the full, unfiltered reality check. The devil is in the details, and Flami is about to show you where all the bodies are buried.

18

Chapter 18: The Devil Is In The Details - A Reality Check

Hi, I'm Flami! Welcome to the breaking ball. Time to get real.

Remember Chapter 0? I warned you.

This book, until now, has been written with a **propagandistic tone**.

Not all statements you've read are 100% true. The devil is in the details.

The technical explanations—bits, algorithms, cryptography, consensus mechanisms—those are accurate. You can trust them, they are trustless.

But the **social implications**, the **promises**, the **revolutionary potential**? Those have been simplified, glossed over, and yes, sometimes exaggerated to keep you engaged.

Why?

Because revolutionary technologies are hard to introduce. If I started with “Bitcoin is slow, expensive, hard to use, environmentally costly, enables crime, can concentrate wealth, and most projects are scams,” you would have stopped reading on page 3.

So I took the role of a “crypto” enthusiast—“cryptobro” some people call them. I distinguish “cryptobro” from “cryptotechbro”: a cryptobro focuses on price speculation, knows nothing or very shallow details on the technology, and has never written nor read any actual source code. A cryptotechbro, on the other hand, understands the technology deeply and can read and write code, but still believes in the hype and propaganda around crypto without spending much time thinking about the social implications. Usually we are nerds, so why would you expect us to think about social interactions? Yeah, the author is one of them making the effort to think about social implications.

Anyway, I showed you the dream. The potential. The best-case scenarios.

Now it's time for the reality check baby.

This chapter will systematically address what I've oversimplified, what I've omitted, and where the propaganda diverges from reality.

Buckle up. This is going to be uncomfortable.

18.1 Part 1: Is Bitcoin Actually “Money”?

18.1.1 What I Told You

“Can we make bits mean ‘money’? The answer is yes.” (Chapter 9)

Bitcoin has the 8 properties of money: scarcity, verifiability, divisibility, etc.

18.1.2 The Reality Check

Bitcoin succeeds at some properties of money, struggles with others.

Let’s revisit those 8 properties with a balanced yin-yang perspective:

1. **Hard to control scarcity:** **Clear success.** 21 million cap enforced by code and social consensus.
2. **Verifiability:** **Clear success.** Anyone can verify transactions mathematically.
3. **No double-spending:** **Clear success.** Blockchain’s core innovation prevents this.
4. **Transferability:** **Mixed reality.** - **Yin (challenges):** Eventually requires internet, electricity, and technical knowledge. Not as simple as handing someone cash. Geographic barriers remain (internet access, regulatory restrictions). - **Yang (potential):** Technical complexity is actively being reduced through Lightning Network, better wallets, and improved UX (user experience). This is not the hardest problem the industry has solved—it’s solvable with time and engineering effort. Furthermore, Bitcoin is just one specific CDN; the benefits of the technology do not vanish because one specific implementation struggles. It would be like saying that if Toyota goes bankrupt, the whole automobile industry will collapse.
5. **Ownership:** **Conditional and complex.** - **Yin:** You own Bitcoin IF you control private keys, understand security, don’t lose seed phrases, and don’t fall for scams. Analysts estimate that roughly **10–20%** of all bitcoin may be permanently lost (recent analyses suggest between 2.3–4 million BTC lost). Even if you have gold in your home but your house is easily burgled, do you really “own” it? - **Yang:** True ownership without intermediaries is possible for those willing to learn. No bank or government can seize properly-secured Bitcoin. Effective self-sovereignty has a learning curve, but it’s genuinely achievable—it is not even close to learning rocket science.
6. **Fungibility:** **Theoretical yes, practical challenges.** - **Yin:** Coins from hacks/crimes (“tainted coins”) can trade at discounts. Address tracking links transaction histories. Some exchanges reject “dirty” Bitcoin. Privacy is harder than it seems. - **Yang:** Value is value. Even “black market” fiat money gets laundered into clean currency—ethical claims on money are as fragile as humans’ ethics themselves. Tainted BTC will likely, as has always happened throughout history, eventually will likely return to licit transactions through modern forms of laundering.
7. **Divisibility:** **Clear success.** Down to satoshis (0.00000001 BTC)—far better divisibility than gold or cash.
8. **Durability:** **Context-dependent.** - **Yin:** Bitcoin requires active infrastructure (nodes, internet, electricity). In civilization collapse or internet failure, Bitcoin stops working. Gold is more durable in cataclysmic events. Bitcoin can’t function offline like some programmable CDNs (Ethereum can temporarily work via signature exchanges even without internet—the “checks” analogy from earlier chapters). - **Yang:** In doomsday scenarios, will we really care about gold either?

If peace returns, databases with BTC data could be restored and the network restarted. For 99.99% of realistic scenarios (not apocalypse), Bitcoin’s digital durability (copies everywhere) is actually superior to physical assets (which can be destroyed, seized, or degraded).

9. Extra: Portability Physically moving gold is hard, or big piles of cash. However, any amount of bitcoin can be moved with you with a small paper or even nothing at all if you manage to remember 12 words in order inside your brain.

18.1.3 Additional properties Bitcoin DOESN’T fully have (yet):

9. Unit of Account: **Mostly absent today, potentially changing.** - **Yin:** Almost no one prices goods in Bitcoin. Prices are set in fiat, converted to BTC at transaction time. This creates accounting complexity and volatility risk. - **Yang:** This is ultimately a choice. More people can slowly choose to display prices in BTC in their stores. El Salvador made BTC legal tender—showing it’s possible. As volatility decreases with maturity, pricing in BTC becomes more viable.

10. Medium of Exchange: **Limited today, situationally useful.** - **Yin:** Very few merchants accept Bitcoin. Those who do often convert immediately to fiat. Volatility makes it terrible for daily transactions in stable economies. - **Yang:** In countries with hyperinflationary currencies (Venezuela, Argentina, Zimbabwe), Bitcoin is better than local money. Besides, even if not “cash,” it can be used gold-like as a store and transfer of value. Even if it’s not a realistic everyday money for most countries, it’s still an economically useful asset.

11. Store of Value: **Volatile but improving.** - **Yin:** Bitcoin has had multiple 70-80% drawdowns. It’s more like a speculative tech stock than “digital gold”—at least for now. - **Yang:** As markets mature, more people adopt, and liquidity increases, volatility tends to decrease (see: gold, equities, real estate over centuries). If adoption keeps growing, Bitcoin’s volatility will likely continue declining. This is just how markets behave.

18.1.4 The Balanced Truth

Yin: Today, Bitcoin is primarily used as a **speculative investment**, not as money. Most economists and central banks do not consider it money, at least they don’t say it publicly, in the functional sense.

Yang: This is ultimately a matter of adoption and choice. If people agree on gold, they can also agree on Bitcoin—there is nothing fundamentally different about that mechanic. In fact, if we had a high liquidity Bitcoin market, its properties make it better and more comfortable to use than gold. Bitcoin is just one specific CDN implementation; CDNs (consensus-based database networks) are far deeper and more useful than just Bitcoin. As the saying goes: “Bitcoin is not crypto, and crypto is not Bitcoin.” The technology has broader applications, and Bitcoin’s current limitations don’t define the entire field’s potential.

18.2 Part 2: The Centralization Problem

18.2.1 What I Told You

“Everyone has a copy of the blockchain. No single person controls it.” (Chapter 9)

Bitcoin is decentralized. No central authority.

18.2.2 The Reality Check

Bitcoin’s power is concentrated in practice, but less so than traditional systems.

18.2.3 Mining concentration:

- **Yin:** Recent pool statistics show the top 4 Bitcoin mining pools control **on the order of 60–70% of hashrate**, though this fluctuates (for example: Foundry ~32%, AntPool ~19%, F2Pool ~8%, SpiderPool ~6%). Top 10 pools control ~90%. If these pools colluded (or were coerced by governments), they could censor transactions, double-spend, or halt the network. This is a real centralization risk.
 - **Yang:** Any miner can leave an unfair pool and join another. In practice, switching costs, loyalty, and incentives keep miners concentrated—but the option exists. The path to further decentralization is clear: more pools and more miners. Running a full node costs approximately \$500-800 in hardware plus electricity. If you have capital, you can become a miner or gather investors to create a new pool. A decentralized oligarchy of many small groups is structurally better than a dictatorship. And crucially: an oligarchy of miners is not the end of the story. What Bitcoin *is* gets ultimately decided by ALL nodes, even the cheap ones. A more decentralized mining landscape with more oligarchies strengthens the network against external attacks.
-

18.2.4 Geographic concentration:

- **Yin:** Most Bitcoin mining happens in a few regions with cheap electricity (historically China, now USA, Kazakhstan, Russia). Governments CAN attack Bitcoin by targeting these concentrated facilities. China’s 2021 mining ban reduced global hashrate by ~50% temporarily—proving this centralization risk is real.
 - **Yang:** Major mining regions (USA, Russia, China, Kazakhstan) are geopolitically independent—often adversarial—actors. They don’t trust each other by nature, which is precisely the problem CDNs came to address. The geographic distribution of mining across rival nations actually provides resilience: no single government can unilaterally shut down the network.
-

18.2.5 Wealth concentration:

- **Yin:** Top 2% of Bitcoin addresses hold ~95% of all Bitcoin. Early adopters and insiders became extraordinarily wealthy through information asymmetry. Bitcoin's wealth inequality (Gini coefficient) is WORSE than traditional financial systems. This creates economic power imbalances.
 - **Yang:** Nothing new under the sun. Early internet adopters also got extraordinarily rich (Google, Amazon, Microsoft, Meta/Facebook are now the world's most valuable companies). It's valid to criticize wealth concentration, but this doesn't uncover any "new evil"—it's human nature and timing. If your political system is easily corruptible by money, is that Bitcoin's fault? Should we deny technological improvement, or should we demand better institutional legitimacy to keep humanity moving forward? If we fix institutional corruption, wealth inequality becomes a manageable policy question, not an existential threat. Also worth noting: in Bitcoin, wealth does not equal power over the protocol. Mining hashrate and node operation determine network rules, not coin ownership. You can own zero BTC and still run a node that validates the rules.
-

18.2.6 Ethereum's staking concentration:

Validators are the analogous to miners in the CDN of Ethereum. They do a thing called staking to prove their worth for writing to the database.

- **Yin:** Post-merge, major staking providers (Lido, Coinbase, Kraken) control the majority of staked ETH. Lido historically peaked around **30–32%** of staked ETH and as of late 2025/early 2026 sits in the **mid-20s%** range (around **24–30%**). Validator sets are even more concentrated than Bitcoin mining. One entity controlling nearly a third is concerning. The iron law of oligarchies keeps appearing—can we actually create a CDN that is truly decentralized?
- **Yang:** ETH is just another CDN, not Bitcoin—so again this doesn't condemn the entire field. To defend ETH specifically: Lido is a way to *aggregate* individual stakers' ETH—similar to mining pools but actually easier to leave. Individuals stake *through* Lido; they can withdraw and switch to other providers like Rocketpool. Developing staking software for Ethereum is relatively cheap, and moving funds around for this use case is absurdly cheap onchain. The Ethereum community actively advertises decentralization and warns when certain clients or staking platforms become too dominant—the community cares and responds. People who stake already demonstrate high knowledge and care—they're more likely to act responsibly. The ecosystem is self-correcting through social pressure and education.



Figure 18.1: Ethereum client diversity warning

In the image above we can see how a software that runs Ethereum, Lighthouse, is becoming too predominant and there are warnings to stop using it as much. As a curious fact, the author works for the very same company that develops the Lighthouse software.

Furthermore, regarding oligarchies, that is where the magic of CDNs comes in. As we said, with Bitcoin, an oligarchy of miners does not destroy the value of the network because ultimately the data is the final source of truth, and that is distributed across all nodes—which are far cheaper to run and actually very decentralized.

18.2.7 Development concentration:

- **Yin:** Small groups of core developers control protocol evolution. Bitcoin Core team and Ethereum Foundation wield enormous influence. Users technically can reject updates, but practically they follow developer recommendations. This creates “benevolent dictator” dynamics.
- **Yang:** Users CAN reject updates—this isn’t theoretical. Developers who care about the code can increase in number; it’s open source. If people start caring more, more developers will join and write alternative implementations to decrease dependency. This concern is most valid while “crypto” isn’t widely adopted. But if/when adoption grows, it will make sense for nations to use taxpayer money to pay excellent developers to write their own Bitcoin/Ethereum clients, minimizing dependency on any single team. If the community grows with awareness, this risk will be significantly mitigated. And crucially: this doesn’t break

CDNs right now. If something fails, battle-tested software that's been running for years is already available. Nodes can just quickly switch to an older version or other software. The system is resilient to any single team disappearing.

18.2.8 The Balanced Truth

“**Everyone has a copy**” is technically true but misleading. **Power is NOT equally distributed.** Computational power and wealth do concentrate control over writing future data. But it does not grant extra influence on what the current data actually is, as that is stored by practically anyone.

The system is **way less centralized than a bank**, but **far from being at the ideal of all actions being fully decentralized**.

- **Yin:** Bitcoin has real centralization risks that need ongoing attention and mitigation.
 - **Yang:** “Way less centralized than a bank” is a significant improvement—a good direction. The system has mechanisms for decentralization (forks, alternative clients, geographical distribution, permissionless participation) that traditional systems lack. Perfect decentralization may be impossible, but meaningful decentralization—enough to resist single-point-of-failure, completely arbitrary censorship—is achievable and actively maintained. These technologies have done nothing other than improve thanks to these traits.
-

18.3 Part 3: The Bitcoin Energy Problem

18.3.1 What I Told You

“Using energy as a scarce resource aligns incentives. Whether that trade-off is worth it is for society to decide.” (Chapter 10)

18.3.2 The Reality Check

18.3.2.1 Yin

Bitcoin’s energy consumption is massive and growing.

The scale: - Bitcoin uses more electricity than entire countries (comparable to Argentina, Netherlands) - The Bitcoin network consumes on the order of **140–170 TWh per year** (recent estimates cluster around this range; ~150 TWh is a reasonable mid-range figure) - Estimates for energy per **on-chain** Bitcoin transaction vary widely, but many put it in the **hundreds to over a thousand kWh** range (e.g., 700–1,400+ kWh depending on methodology) - Carbon footprint depends on energy mix (often fossil fuels in regions with cheap power)

The price-hashrate feedback loop: - “Bitcoin is valuable, so energy expenditure is justified” - But Bitcoin is valuable largely due to speculation, not utility today - Energy expenditure is proportional to **hashrate competition**, which increases when mining is profitable - When Bitcoin price doubles, mining becomes more profitable, more miners join, hashrate increases, and energy consumption increases - This happens even if transaction volume stays flat—the energy isn’t serving more users, just securing higher speculative value

The environmental externalities: - E-waste from obsolete mining hardware (ASICs become obsolete every 1-2 years) - Mining concentrates in regions with lax environmental regulations - Fossil fuel use is common (cheap energy often means coal or natural gas) - These costs are externalized (society pays through environmental damage, miners profit)

The comparison problem: - “Banks use energy too!” — True, but banks serve billions of users daily with diverse services. Bitcoin primarily serves millions of speculators. - “Gold mining uses energy!” — Gold has industrial uses (electronics, dentistry) and aesthetic/cultural value spanning millennia. Bitcoin’s primary use today is speculation. - These comparisons are whataboutism, not justification for the energy expenditure.

“**Society decides**” is misleading. Society doesn’t meaningfully decide Bitcoin’s energy use—it’s determined by miners maximizing profit within competitive dynamics. There’s no democratic process, no vote, no collective decision-making mechanism.

Proof-of-Stake addresses this (Ethereum uses 99.95% less energy), but introduces different security and centralization tradeoffs.

18.3.2.2 Yang

Energy sources are a general humanity problem, not Bitcoin-specific.

The main issue—carbon-emitting energy sources—is a civilization-wide challenge affecting all industries. Bitcoin doesn’t create this problem; it inherits the existing energy infrastructure. As renewable energy becomes cheaper (which it is—solar and wind are now the cheapest electricity sources in most regions), miners naturally migrate to these sources for profit maximization. Blaming only Bitcoin for this ignores the actual problem: our global energy mix.

E-waste is a broader technological challenge too:

Hardware obsolescence isn’t unique to Bitcoin. Old ASICs can be repurposed for other compute tasks: personal computers, gaming PCs, training local AI models. Yes, this is a concern—but it’s part of the larger e-waste problem. How many Apple phones are thrown away each year? How many new iPhone versions launch with questionable improvements? E-waste is a real concern across all technology, not just Bitcoin. We should address it systematically, not mention just one technology.

Against the speculation criticism:

Yes, Bitcoin is primarily speculative today—but the reader should understand by now that **this is a choice**. You can help change it. If you’re a speculator, you can shift your framing: “I’m going to hold Bitcoin because I agree it has value for coordination, not just profit.” Everyone’s voice matters. Value is social consensus, and you’re part of that consensus. Be one more voice pushing for real gold-like utility, not just speculation.

Cost-benefit analysis is context-dependent:

You might say, regarding the CO₂ emissions and other monetary costs... okay, planes pollute, but at least they are blatantly useful.

Are Bitcoin’s costs worth the benefit? We’ll only see when people use it at its fullest potential—as a new and better form of digital gold, as a neutral settlement layer for international agreements. Programmable international coordination can bring enormous value. We can even say, without

exaggeration, that better coordination mechanisms can save lives (preventing conflicts through economic interdependence). Is every human life saved worth the cost of running CDNs?

The problem with assessing the cost-benefit of CDNs like Bitcoin is that we would be trying to measure the value of generating a revolution in how we trust—and that is unmeasurable because the way people, entities, and nations talk to each other would fundamentally change. The potential is clearly enormous and hard to quantify. Just like with the current developments in Artificial Intelligence.

This debate is eternal and normal:

The benefits of certain CDNs like Bitcoin relative to their costs is a true, ongoing debate. Just be fully aware of both sides. This debate will never end—it's inherently human: “Should we keep using tool X? Should we switch to tool Y?” Ultimately, this is classic human coordination, nothing special to Bitcoin. It happens with all products. Trade-offs are evaluated, society chooses, society tries the new technology, and if the choice brings an overall benefit, it gets adopted.

The main issue with CDNs like Bitcoin is that the benefits are pretty much impossible to measure quickly. We would need to use the technology for one or two generations and see if the programmable economic dependency correlated with a decrease of war threats or even wars themselves and a betterment in quality of lives globally.

18.4 Part 4: The Exaggerations

18.4.1 “Prove ownership of digital assets without central registry”:

Yin : You can't really separate ownership from the physical world.

At the end of the day, if your house is your house, it's because if someone steals it from you—like enters to live there and says it's theirs—the only thing that will make you regain ownership is showing the police that in the State's database or records it says it's yours.

Cryptographic ownership does not exist legally as of now. Most people use custodial services (Coinbase, etc.), reintroducing intermediaries. Legal recognition remains unclear—courts may not enforce cryptographic ownership without legal identity.

Yang : As adoption grows, parliaments will be pressured to create legal frameworks recognizing cryptographic ownership. What we have now are better, untamperable records—you can see what's happening with your property and detect immediately if something sketchy happens to the ownership records. That's the real innovation.

Bottom line: CDNs don't come here to guarantee physical ownership. They come to provide transparent, untamperable records that can support ownership systems.

18.4.2 “Coordinate without trusted intermediaries”:

Yin : The claim of no intermediaries is false. You're still trusting the nodes.

Yang : The nodes are so many and so well designed that trust is practically guaranteed through decentralization. This is where the new word “trustless” comes from—not that you trust nothing, but that you trust a system designed so no single party has a rational incentive to ever betray you. The only real threat is external groups trying to break the system from outside or infiltrate spies to break it from within.

18.4.3 “Physically impossible things are now possible”:

Yin : “Physically impossible” is an exaggeration.

Yang : No new physics were invented—only clever uses of math and computers (new engineering feats). So more accurately: **technically impossible** before the breakthrough. But if no one knew how to build it, even though the materials existed, it was **practically impossible** to have it—which, for all intents and purposes, made it feel “physically impossible.”

18.5 Part 5: Smart Contracts Aren’t That Smart Nor Unstoppable

18.5.1 What I Told You

“Smart contracts are unstoppable. No one can change the rules once deployed. Code executes automatically.” (Chapter 13)

18.5.2 The Reality Check

Admin keys and upgrade mechanisms:

- **Yin:** Some deployed contracts include “owner” addresses with special privileges. Common admin powers: pause contract, upgrade logic, mint tokens, adjust parameters. Many DeFi (Decentralized Finance) “hacks” involve compromised admin keys or malicious developers using their privileges. “Unstoppable by default” should be “stoppable by default, with rare exceptions.
 - **Yang:** Anyone can create the very same version of code but without an owner. Users are completely free to interact with whichever code they see fit. And, by default, smart contracts do not have an owner. It is something a human has to code explicitly.
-

Oracle dependencies:

- **Yin:** Smart contracts need external data (prices, weather, sports scores). Oracles are centralized points of failure. If an oracle is manipulated, the contract executes incorrectly despite code being “correctly programmed.” Trusting oracles reintroduces trust assumptions.
- **Yang:** Innovation is ongoing—CDNs completely specialized in providing data (Chainlink is an example) are being developed. Designing trust-minimized oracles with CDNs and game

theory alignment is an active area of research and development, already being used to safely handle millions of dollars of value.

Gas costs:

- **Yin:** Deploying contracts costs hundreds to thousands of dollars when Ethereum is contested. “Anyone can deploy” is technically true, practically false for most people. Interacting with contracts also costs gas—complex DeFi operations can cost \$50-200 in fees.
 - **Yang:** Layer 2 dramatically reduces costs. Open source software to ease interaction is emerging, along with tutorials across the internet. It’s true that not anyone right now is able to deploy their own bank with no computer science knowledge and 1 click—but that’s doable. We just need 2 or 3 nerds to code it as a summer project. And that’s not unrealistic nerd behavior at all. Furthermore, L1s are also becoming more and more cheap to use as time goes and technological improvements are created.
-

Code transparency user understanding:

- **Yin:** “Transparent and auditable” assumes users can read Solidity code. 99.9% of users cannot audit smart contracts. Even audited contracts get exploited (see: dozens of audited DeFi hacks). Transparency benefits sophisticated attackers more than average users.
 - **Yang:** The security industry is very profitable and needed, therefore growing. The oligarchy of code verifiers is growing and becoming more decentralized. Ways to seamlessly pay insurance in case of a hack are being developed. The ecosystem is maturing and smart contracts are growing more secure every year. Security is still a big concern that should not be ignored, but the trend is clear and positive.
-

The reality: Smart contracts are powerful, but they’re not magic. They’re code running on distributed systems which, even if the system they run on is trustless, the contracts can have code programmed on them with trust assumptions, centralization points, and different social governance.

They are still not secure enough for most people to blindly trust with large amounts of money without due diligence. Yet, this trend is improving over time and it is not that big of a challenge to overcome. As you can see, there are job opportunities everywhere for hackers willing to help secure the smart contract and CDN ecosystem of the future. The author is one of those hackers who is day by day helping secure this ecosystem.

18.6 Part 6: Layer 2s Don’t Solve Everything

18.6.1 What I Told You

“Layer 2s enable fast, cheap transactions while Layer 1 provides security. The best of both worlds.” (Chapter 15)

18.6.2 The Reality Check

Security is NOT Layer 1 security:

- **Yin:** There are lots of technical designs for L2s, each with their own set of challenges, trade-offs, and complexities. Some might not be examined as thoroughly by security researchers (hackers who protect code instead of abusing it), and therefore they might have vulnerabilities not yet discovered. The ecosystem is still young and maturing.
 - **Yang:** All systems start young and fragile. L2s are even younger than Bitcoin. This is a very ambitious system, therefore it will take time to mature. But the trend is clear: more people are working on it, more money is being invested, more knowledge is being accumulated. The ecosystem is maturing.
-

L1 checkpoints don't guarantee complete safety:

- **Yin:** If L2 operators don't post updates to L1, there's no "checkpoint." L1 can't fully validate L2 logic—it just stores summaries. If L2 operators collude and post false information, L1 won't detect it unless fraud detection mechanisms work. Many L2s have centralized controllers that can block transactions before they reach L1.
 - **Yang:** L2s are improving their security models and becoming more decentralized, putting effort into avoiding single points of failure. For example, the decision of posting data to L1 to create a checkpoint doesn't rely only on one entity. L2Beat is a website where criticism is made of every Layer 2 to warn about their centralization points. The whole ecosystem is aware of this and applies constant pressure and criticism to these L2s to improve. Furthermore, novel engineering techniques are being developed to avoid these kinds of risks. Overall, the ecosystem is maturing and Layer 2s are slowly becoming more trustless.
-

The UX (User Experience) complexity:

- **Yin:** Users must understand technical concepts, manage multiple wallets, trust bridge contracts (many hacked for hundreds of millions), and pay multiple transaction fees. "Give us time and we'll simplify it" has been the mantra for years with limited progress. The complexity isn't just UI (User Interface) polish—it's systemic (multiple trust models, different security guarantees, fragmented liquidity).
 - **Yang:** Even though "we will simplify it" has been going on without completion, it's because there are harder problems to prioritize. Solving UI/UX problems is not the hardest problem this industry faces right now. Current efforts are proposing standards that make it easier at a technical level to interact with multiple databases (chains), and their creation will lead to UX simplification too. Furthermore, it's just a matter of time and more people working to fix the UX problems. The industry is actually young and small.
-

Network effects fragment:

- **Yin:** Liquidity splits across L2s (your money on one L2, your friend's on another = hard to transact). Developer attention fragments (build on which L2?). User confusion (which L2

should I use?). The “societies within societies” framing is pretty, but it’s coordination chaos in practice.

- **Yang:** Ways to create code that works across any chain (“write once, use anywhere”) are being developed. User confusion will disappear when UX improves—they’ll just see their money, their value, and different investment options. Advanced users will be able to click details to see which database they’re in and its risks. But for now, that’s not the case. Again, yes, now it’s chaos and messy, but the ecosystem is maturing and these problems are being addressed.
-

Linux comparison and adoption:

- **Yin:** I said “Linux works perfectly and anyone can use it.” Reality: Linux desktop share is still a **small single-digit percentage**, but has recently climbed to around **4–5%** globally (as of 2025, up from <3% in previous years). Linux succeeds in servers (managed by professionals) but still struggles in consumer adoption. Using Linux as proof that “anyone can use complex open-source software” contradicts decades of evidence. **“Give us time and we’ll give you collective freedom”** is hopium. After 15+ years, the problems haven’t decreased—they’ve increased.
 - **Yang:** People CAN learn to use Linux if they put in a few afternoons of effort—5 days at most. It’s not that complex, but people’s attention is a scarce resource. We must make efforts to get people to care about this, or make it so simple that people don’t need to care about it. Both paths are being worked on. If you invest 5 days of your life learning a new tool that can grant you more power and freedom, over time you might even recover them because of the more efficient society you live in—you might get 6 days of holidays down the line. Sure, it’s a big effort and must be done by lots of people, so until everyone does it, the benefit likely won’t be seen by most people. Things that are worth it require effort: learn and teach, or make it even easier for others to learn. The “laziness” and ignorance of the masses about what they should prioritize learning in their lives is a problem—but it can be overcome. People wait to cross streets when there’s a crosswalk; people learned to use smartphones—which might feel easy now but isn’t. It’s a massive change we made (see an old person trying to learn it). This one isn’t that massive; we just need teachers, and preachers even, but mainly friendly teachers. Selfless teachers if possible.
-

18.7 Part 7: The No Trust Myth

18.7.1 What I Told You

“You don’t need to trust anyone. Math doesn’t lie. Coordination through math, cryptography, and incentives.” (Chapter 17)

18.7.2 The Reality Check

“Trustless”, even if novel and necessary, is still a bit of a misleading term. You still trust many things.

What you're trusting:

1. Cryptographic assumptions:

- **Yin:** Assume elliptic curves aren't broken (quantum computers might break them). Assume hash functions are secure (SHA-256 collision resistance). Assume no mathematical breakthroughs break current crypto.
 - **Yang:** To be fair, the whole world works under this assumption. Current banking, military systems, your phone—all modern cryptography relies on these assumptions. Therefore, it's not a new problem unique to CDNs or even to humanity. Yes, if a brilliant mathematician figures out a way to break this tomorrow the world might become chaos, sorry for revealing you this assumption under which we all live with.
-

2. Code implementations:

- **Yin:** Trust developers wrote error-free code (they don't—smart contract hacks exist). Trust no backdoors were introduced (can you audit millions of lines of code?)
 - **Yang:** Error-free code, not only in CDNs but in general, is impossible—that is true. But specialists work every day to make sure the bugs are small and not impactful. Join them if you please, but it is very difficult to master. The code is getting more and more secure. Everyone uses and will use this software, so it will be in the interest of all nations to get it right and make sure no other nations put weird code that can be used against them. Therefore, there is a massive incentive to get it right. AAVE, a “bank on the blockchain,” is handling already billions of dollars in value from people all over the world. The idea of code containing bugs and backdoors is a valid concern, yet it is extremely complex to actually exploit. Only people with lots of time and lots of knowledge can do it, like geniuses financed by nation-states. Spying software in everyday code used in devices is a thing because the code is private and controlled by corporations which might be bribed or just fooled. But this time, the code is public infrastructure, therefore all nation-states that are against each other will make sure no one introduces “hacks.” Sure they will try, but others will be watching and notifying. Now it is just a “few billion” dollars on the line, but in the future, if the majority of the world's finance moves to CDNs, the incentive to get it right will be enormous for anyone.
-

3. Economic incentives:

- **Yin:** Trust that miners/validators remain economically rational. Trust that 51% of hash-power stays honest in the case of Bitcoin. Trust that incentives continue aligning—what if economics change?
 - **Yang:** Current CDNs are clearly showing that the consensus algorithms are well designed. It can happen that a newly created one does not get it right, true, but knowledge on how to do it is already established. Only external actors can take down these systems, whether from external attacks or infiltrating from within. The economic incentives are well designed and they are proven to work. The real worry you should have is external attackers and the new incentive designs in new CDNs who have been running for not that much time.
-

4. Social consensus:

- **Yin:** Trust that community won't hard fork against your interests. Trust that developers won't introduce changes you dislike. Trust that users won't abandon the network (causing collapse in value).
 - **Yang:** This is an inherent human nature problem, not really specific to CDNs. Every day you wake up you trust the internet will work, that no one will do a coup d'état in your country, that the parliament won't pass a law that prejudices you. These are problems arising from social coordination and consensus—very old problems with very old dynamics. If these things happen, you either change networks or adapt. Sure, be aware of newly created CDNs that are especially centralized; the risk is real, the same way you have to be careful if a State is a dictatorship and one day the dictator decides to act against you. The best thing to do to assure peace of mind is to see which CDNs have the most reliable track record of not altering their consensus frequently or quickly or arbitrarily.
-

5. Infrastructure:

- **Yin:** Trust that internet remains available. Trust that electricity grid works. Trust that nodes continue running. Trust that exchanges/wallets provide access.
 - **Yang:** Again, the whole world relies on this today. Regarding nodes, it is like saying trust that your router device at home will work. If a CDN has few nodes, it has more chances of being taken down accidentally or maliciously—so take that into account. For now, big CDNs almost never experience this. Ethereum has had **near-perfect uptime** since 2015, with no major chain-wide outages (celebrating 10+ years with effectively no network-wide downtime). Bitcoin has achieved ~**99.99% uptime** since 2009, with only two significant incidents early in its history. That mimics current internet technologies' uptime. Centralized exchanges only make sense in the world of today, but in the future when everyone is onchain they will stop making sense. DEXes (decentralized exchanges) already exist and you don't need permission from anyone to use them. Furthermore, lots of wallet software exists, even open source, and even more is being created. The likelihood of you not being able to access your money because of a wallet problem is ridiculously low. JUST REMEMBER: NOT YOUR KEYS, NOT YOUR COINS. A wallet stops working? You use another one, activate your keys, and move the coins.
-

6. Oracles and bridges:

- **Yin:** Most of this new decentralized finance (DeFi) relies on price oracles (Chainlink, etc.)—centralized trust points. Cross-chain bridges require trusting bridge contracts (many hacked).
 - **Yang:** As said before, lots of innovations are being made to make oracles and bridges more trustless. The ecosystem is maturing and these problems are being addressed. Chainlink has been working for years now providing reliable and accurate prices and more world data to these networks, and there are more oracles. The systems do work in a trust-minimized way. If you are curious, they also usually run local CDNs with their classic economic incentives guaranteed by game theory and cryptography to make sure no one from within cheats.
-

Conclusion:

Indeed the accurate term is “trust-minimized” (reducing trust requirements), not “trustless” (which seems to imply eliminating trust completely). But to be honest, all human interaction is eventually based on trust. Trustless is impossible for human nature. The idea of designing technological systems that systemically reduce trust-needing factors is still new and should have a word. You choose it to be honest—the industry is new and the term that will stay is the one people eventually use.

I prefer trust-minimized. Which one do you?

18.8 Part 8: The Use Case Problem

18.8.1 The Elephant in the Room

After 15+ years, **crypto’s primary use case is speculation**, not the revolutionary applications promised.

What people actually use crypto for:

1. **Speculation/Investment (90%+ of volume):** - Buying/holding Bitcoin hoping price goes up, not using it as money - Trading fancy volatile coins (like the so-called Meme coins) to make quick profit and leave
2. **Illicit activity (significant but declining %):** - Ransomware payments - Dark web markets - Money laundering - Tax evasion - Sanctions evasion - Ponzi schemes that are clear scams
3. **Actual utility (~1%):** - International remittances (some use, but fees + volatility limit adoption) - Censorship resistance (activists in authoritarian regimes—real but tiny volume) - Unbanked financial access (mostly aspirational, limited real-world success)

What crypto is NOT widely used for: - Directly buying goods/services (virtually zero merchant adoption) - Paying employees (rare, mostly crypto companies) - Storing savings (too volatile or risky due to hacks) - Traditional banking functions (lending, mortgages, etc.)

The gap between promise and reality is enormous.

The book presented use cases as if they’re current reality. Most are aspirational at best, fantasy at worst.

Yin: After 15+ years, the primary use remains speculation. The revolutionary applications haven’t materialized at scale. Most promises remain aspirational. Illicit activities represent significant usage, which raises serious concerns about enabling crime.

Yang: The change this technology brings is deep—redefining trust architecture is a colossal civilizational change involving everyone from citizens, to developers, to governments. This will take generations, not just a few years.

On illicit activities: Every transformative technology gets weaponized initially. Guns, the internet, encryption—all enabled more sophisticated ways of doing crime. The deep web facilitates illegal markets, yet we didn’t abandon the internet. We learned to manage risks while capturing benefits. Most people use the internet legally today. CDNs will follow the same pattern: society will develop frameworks to mitigate abuse while preserving the advantages.

On timeline expectations: Changing institutions, habits, and collective understanding requires time and collective effort. If you expect the classic developer mindset of “move fast, break things” progress, you’ll be frustrated. Trust-minimization technology demands careful development, education, marketing, and social consensus-building across all of society—from the most democratic to the most authoritarian nations.

On the speculation-to-utility transition: The current speculative activity, while not the end goal, is funding infrastructure development. Early internet companies were also built on speculative investment before proving utility. The difference between speculation funding development and speculation as the only purpose is that development is actually happening: Layer 2s are being built, wallet UX is improving, regulatory frameworks are emerging. Whether this development leads to mainstream utility in 5, 10, or 30 years remains uncertain—but the work continues.

Be patient, resist gambling temptation, spread knowledge, and help build this technology. Yes, it’s difficult to not sound like a sect when building revolutionary technology—I’m sorry. You can do more than this, more in the following section.

18.9 Part 9: The Ideology Problem

18.9.1 What I Told You

“Bitcoin doesn’t have a political opinion. It’s neutral technology. Humans give it meaning.” (Chapter 17)

18.9.2 The Reality Check

Yin: I told you this for the sake of simplicity, marketing, and reader engagement. But it’s misleading. CDNs are built on social consensus, therefore they embed the ideologies of their communities.

The technology layer itself may be neutral, but **how we apply it and what values we encode into it are not.**

Bitcoin is deeply ideological. Bitcoin is NOT neutral—it’s opinionated technology, sustained by a community which values specific principles like limited supply, censorship resistance, and anti-inflationary policy.

Yang: Yet this doesn’t mean the technology is inherently evil or broken. All human institutions embed values—that’s unavoidable. What matters is: 1. **Transparency:** Bitcoin’s values are explicit and visible in its code 2. **Choice:** You can fork it or create alternatives with different values 3. **Participation:** Anyone can join the conversation about what those values should be

Traditional financial systems also embed ideologies: Keynesian economics, central bank control, surveillance... And you can not escape them as easily at all.

18.10 Part 10: So What Should You Do?

You should have enough knowledge to think on your own by now, but here's practical advice:

- 1. Only risk what you can afford to lose.** - Crypto is highly speculative and volatile today - Don't use crypto for savings—use stable, boring traditional instruments instead - Treat it like lottery tickets or casino chips, without gambling
 - 2. NEVER share seed phrases.** - If someone asks for your seed phrase, they're scamming you. Even if it literally is the real CEO of a reputable company. - Not your keys, not your coins.
 - 3. If you experiment, stick to established CDNs.** - Bitcoin or Ethereum are least scammy (still risky) - Avoid altcoins, meme coins, new tokens (99.99% of them are scams whether their creators know it or not)
 - 4. Be deeply skeptical.** - Most crypto projects are scams, rug pulls, or Ponzi's - "Too good to be true" yields? 986% APY on my crypto-dollars? It might disappear tomorrow - Celebrity endorsements? Usually paid promotions in which the celebrity, or their family to make it more discreet, get a portion on a zero-sum game algorithm moving the price - If you don't understand the technology, don't invest - Complexity and weird wording, or overwording something, often hides scams
 - 5. Use traditional finance for most things now.** - Banks, credit cards, regulated brokers have problems but also protections - They work better for most people, most of the time - If you are uneducated or cannot self-control, protection is a good thing actually. Good news is that those two traits can be improved with time and effort
 - 6. Yet, slowly adopt and experiment carefully.** - Accept stablecoin payments in your shop if you own one - Use decentralized banks to earn yield on cash you don't need immediately - VISA/Mastercard already allow indirect crypto spending even if merchants don't accept it - Pay friends for dinners via wallets to get comfortable using them - Keep main net worth out of crypto, but do put some in it to learn. Remember, money you really do not need. - The technology depends on adoption; you're building a more efficient future - Share knowledge with friends—educate without being annoying - Slowly learn to improve the way you store your keys: instead of paper use titanium plates, or encrypt them in multiple USBs, buy a hardware wallet... - Learn to run a node of your favorite CDN and make it even more secure via decentralization.
 - 7. If you believe in the tech, contribute meaningfully.** - Don't just speculate or experiment with spare change - Build, educate, invest in or contribute to the ecosystem - Care about the real potential for humanity
 - 8. Stay humble and keep learning.** - No one knows the future - We may not see massive adoption in our generation - "Number go up forever" isn't a smart investment thesis - Read skeptics AND believers - Form your own conclusions
-

18.11 Wrapping Up The Breaking Ball

CDNs are: - Real technology - Novel capabilities - Overpromising and underdelivering (like all marketers need at some point) - Unsolved problems (like all technologies) - Mostly speculative usage today (you, literally you, can change that) - Crime enablement (like all tools) - Complex to use currently (like all technologies at first) - Uncertain future (like with any innovation)

18.11.1 I like trains.

When trains were invented in the 1800s, imagine telling people: “Trains are slow now, but if we keep investing, they’ll bring you from Tokyo to Osaka in 3 hours instead of 10! We must keep building this ‘train’ technology—it’s clearly the future of transportation.”

Some people would have, understandably, called you crazy or a scammer.

Yet, fast forward through innovations in multiple fields, and today we have high-speed trains that do exactly that.

CDNs are in a similar early phase—the first trains of a trust management revolution. But unlike the 200-year journey from steam engines to bullet trains, we have advantages: global internet coordination, AI-powered information processing, and accumulated engineering knowledge. Huge leaps in innovation can now take just 20 years, maybe less.

18.11.2 The Balanced Truth

For most people, most of the time, traditional systems work better right now.

But for some people in some situations—activists under authoritarian regimes, people in hyperinflationary economies, those building permissionless applications—crypto is genuinely valuable today.

More importantly: The technology is real. The capabilities are novel. The potential remains enormous. Whether that potential gets realized depends on us—builders, educators, critics, and adopters—choosing to work toward the better future while acknowledging and fixing the present problems.

This book showed you the dream. This chapter showed you the reality.

Now you can decide for yourself.

You’ve seen both sides: the revolutionary potential and the harsh limitations. You understand the technology deeply enough to make informed choices. You know the risks and the possibilities.

What you do next is up to you.

Will you build? Will you educate? Will you experiment carefully? Will you wait and watch? Will you critique constructively?

Just do it with clear eyes.

Welcome to the other side of the breaking ball.

— Yours Faithfully, Flami, the realistic flamingo.



Figure 18.2: Flami posing for you

Next: The book is almost done, let's start the conclusion. Chapter 19 will recap the entire journey—from bits to Bitcoin, from cryptography to coordination. You'll see what you've learned and what you can build with that knowledge.

19

Chapter 19: From Bit to Bitcoin - Wrap-Up

“Look how far you’ve come” chapter.

Remember where we started?

A light switch—on or off, 1 or 0. A single bit, the most fundamental unit of digital information.

And now, 18 chapters later, you understand global coordination technology that makes things possible that were technically impossible before 2009.

You built this understanding from first principles.

No jargon handwaving. No “trust me, it’s complicated.” No shortcuts.

We started with a bit and built up, layer by layer, concept by concept, until you intuitively understood Bitcoin, Ethereum, smart contracts, Layer 2s, Zero-Knowledge Proofs, and the philosophical implications of it all.

This is an achievement. Take a moment to appreciate it.

Most people—even smart, educated people—don’t understand this technology. They hear buzzwords, see hype, feel confused, and give up.

But you didn’t. You stuck with it. You learned.

And now you understand way better.

19.1 The Journey

Let’s trace the path we took together.

19.1.1 Part 1: Foundations - What Is Information?

Chapter 1: The Bit - Everything digital is patterns of on/off switches, and computers are billions of tiny transistors working in concert. - All complexity builds from this simple foundation: 0 and 1.

Chapter 2: Logos - Mapping Meaning to Numbers - Humans assign meaning to patterns—ASCII maps 65 to ‘A’, 66 to ‘B’, and so on. - Your name exists in binary, and images, videos, and sound are all just numbers. - **Key insight:** Information is meaning we agree upon.

Chapter 3: Algorithms - Following Rules - Algorithms are just instructions (recipes), and computers follow them perfectly at incredible speed. - They're deterministic: same input leads to same output, always. - **Key insight:** Computers don't "think"—they follow rules.

Chapter 4: Protocols - Computers Talking - A protocol is an agreed-upon set of rules for communication. - HTTP, TCP/IP—the internet is protocols all the way down. - **Key insight:** Protocols are social agreements between machines.

19.1.2 Part 2: Trust & Cryptography - Who Can You Trust?

Chapter 5: The Trust Problem - Alice wants to tell Bob a secret, but Carol is listening on the public channel. - **Key insight:** The internet is public by default.

Chapter 6: Symmetric Encryption - Shared Secrets - The Caesar cipher shifts letters by 3, and modern encryption works similarly: $f(\text{message}, \text{password}) = \text{scrambled_message}$. - **The problem:** How do you share the password without Carol intercepting it?

Chapter 7: Asymmetric Encryption - The Magic Trick - One-way functions are easy in one direction but impossible to reverse. - Two keys work together: a public key (the lock) and a private key (the key). - Digital signatures prove identity without revealing secrets. - **Key insight:** You can prove who you are without giving away your secrets.

Chapter 8: Quantum Computing - The Future Threat? - What breaks: RSA and ECDSA (current cryptographic standards). - What stays secure: Post-quantum cryptography. - Timeline: Quantum computers capable of breaking current cryptography are estimated to always be 10-20 years away. - **Key insight:** The quantum threat is real but manageable if we take it seriously and consistently prepare.

19.1.3 Part 3: Consensus Networks - The Breakthrough

Chapter 9: Money as Synchronized Bits - What makes something money? Eight properties including scarcity, verifiability, and resistance to double-spending. - Bits can have these properties, but the question becomes: where do you store them? - **The insight:** Distribute the database so everyone has a copy. - **The consensus problem:** How do you synchronize without a central authority?

Chapter 10: Proof-of-Work - Earning the Right to Write - Who gets the right to write to the database? You can't just vote—Sybil attacks make that impossible. - **Proof-of-Work:** Solve a computational puzzle by burning electricity to prove you've done real work. - Mining rewards combine new coins with transaction fees, and the incentives align: honesty equals profit. - **Key insight:** Temporary leadership is earned through work and rotates among participants.

Chapter 11: The Blockchain - Chaining History Together - The simultaneous solution problem arises when two miners solve the puzzle at once. - The elegant solution: let them compete, and the longest chain wins. - **The blockchain data structure** makes faking time impossible. - **Proper naming:** Consensus-based Database Networks (CDN) a type of Decentralized Datasync Technology. - **Key insight:** Blockchain is tamper-evident history.

Chapter 12: The Blockchain Data Structure (Optional Technical Deep-Dive) - It's a sort of "linked list made of hash pointers", and its tamper-evident property means changing one block breaks all subsequent blocks. - Why can't you just recompute? Because that requires redoing all

the Proof-of-Work. - The anti-gaslight machine: everyone has a copy, so changes can't be hidden.
 - **Key insight:** The technical details of how blockchain prevents cheating.

19.1.4 Part 4: Evolution & Implications - What This All Enables

Chapter 13: Ethereum - The Consensual Computing Machine - Bitcoin is digital gold with simple transactions, but Ethereum asked: what if the database could run programs?
 - Smart contracts are if-then logic enforced by code, and the Ethereum Virtual Machine (EVM) ensures every node runs the same programs. - Gas explains why computation costs money. - **Key insight:** Bitcoin coordinates on value; Ethereum coordinates on computation.

Chapter 14: When Consensus Forks - The Nature of Agreement - The Ethereum/Ethereum Classic fork followed the DAO hack in 2016. - Bitcoin/Bitcoin Cash split over the block size debate in August 2017. - **The pattern:** Technology enables the fork, humans decide the outcome, and blockchain makes disagreement auditable. - **Key insight:** Forks aren't failures—they're proof that coordination is voluntary.

Chapter 15: Layer 2s and The Trilemma - Societies Within Societies - The Blockchain Trilemma: Decentralization, Security, Scalability—pick 2 out of 3. - Why are CDNs slow? Because every node processes every transaction. - **Layer 2 solutions** are fast, cheap layers that periodically settle with Layer 1. - Examples include Lightning Network for Bitcoin and Rollups for Ethereum.
 - **Key insight:** Consensus can be nested—societies within societies.

Chapter 16: Zero-Knowledge Proofs - Privacy Meets Verification - The problem: CDNs are transparent, meaning everyone sees everything. - The dream: Privacy AND verifiability—which seemed impossible! - **Zero-Knowledge Proofs** let you prove you know claims about X without revealing X. - Three properties define them: Completeness, Soundness, and Zero-Knowledge. - **Key insight:** Zero-knowledge is the missing piece—privacy and verification simultaneously.

Chapter 17: The Philosophy - What This All Means for Humanity - This is coordination technology for strangers at internet scale, making things possible that were technically impossible before 2009. - Power dynamics shift from centralized gatekeepers to distributed coordination. - The anti-gaslight machine means you can't hide tyranny or erase the past (at least economic history). - Honest trade-offs include responsibility, complexity, irreversibility, energy use, and regulation. - **Key insight:** Technology enables, humans decide. This depends on all of us.

Chapter 18: The Reality Check - The Devil Is In The Details - Everything until now was simplified, sometimes exaggerated for engagement. - Bitcoin isn't really “money” for most people yet, mining is concentrated, and energy use is massive. - Smart contracts aren't unstoppable by default, L2s introduce new trade-offs, and “trustless” really means “trust-minimized.” - After 15+ years, the primary use is still speculation—but the technology is real, and change takes generations.
 - **Key insight:** Now you know both sides—the dream AND the reality. No more propaganda.

19.2 Recap: The “Weird” Terms (They Should Feel More Natural Now)

When you started, these words probably seemed like gibberish.

Now they make perfect sense.

19.2.1 Bit

On/off switch - the foundation of all digital information.

You understand: Everything digital—text, images, videos, money—is patterns of bits.

19.2.2 Logos

Humans assign meaning - which is why bits can represent anything.

You understand: Information is meaning we agree upon. 65 = ‘A’ because we say so.

19.2.3 Algorithm

Instructions computers follow - no magic, just rules.

You understand: Computers execute algorithms perfectly, deterministically, billions of times per second.

19.2.4 Protocol

Agreed communication rules - how computers coordinate.

You understand: The internet is protocols. HTTP, TCP/IP, Bitcoin—all protocols.

19.2.5 Hash

One-way fingerprint - can't reverse it, but it proves integrity.

You understand: hash(“hello”) = unique output. Change one letter, completely different hash. Can't go backwards.

19.2.6 Asymmetric Cryptography

Public/private keys - prove identity without revealing secrets.

You understand: Lock (public key) and key (private key). Anyone can lock, only you can unlock. Digital signatures prove you sent a message without revealing your private key.

19.2.7 Consensus

Agreeing on database state - without central authority.

You understand: The core problem Bitcoin solved. How do strangers agree on truth without trusting anyone but the rules?

19.2.8 Proof-of-Work

Effort proves right to write - Sybil-resistant selection.

You understand: Burn electricity to solve puzzle. Winner gets to write next block. Incentives align: honesty = profit.

19.2.9 Blockchain

Tamper-evident history - the anti-gaslight machine.

You understand: Linked list with hash pointers. Change one block, break the chain. Can't rewrite history without redoing all the work.

19.2.10 Decentralized Datasync Technology and CDNs

The better names - synchronizing data across strangers.

You understand: “Blockchain” is just the data structure. The real innovation is consensus-based database networks that coordinate without central authority.

19.2.11 Database and network

Storing and sharing data - the coordinated ledger.

You understand: A database is a structured data storage. A network connects computers. A CDN is a database shared across a network of computers which requires a consensus for its data to be altered.

19.2.12 Smart Contracts

Programs that modify the data on CDNs - if-then logic that can be unstoppable.

You understand: Code that executes automatically on distributed systems. No intermediary needed, though many contracts have admin keys. If conditions met, then execute. Powerful but not magic.

19.2.13 Forks

Consensus splits - proof that coordination is voluntary.

You understand: When communities disagree, they can split. Both chains exist. Market decides value. Disagreement is auditable forever.

19.2.14 Layer 2

Societies within societies - nested coordination.

You understand: Fast, cheap layers that periodically settle with slow, secure Layer 1. Like states (L2) and federal government (L1).

19.2.15 Zero-Knowledge

Prove without revealing - privacy and verification simultaneously.

You understand: Mathematical magic. Prove you know something without revealing what you know. Completeness, Soundness, Zero-Knowledge.

19.3 What You Now Understand

You understand more than just the technical concepts.

You understand how strangers coordinate with minimized trust: - Math provides verifiable truth. - Cryptography enables secure communication and identity. - Incentives align selfish actors toward collective benefit. - Social consensus decides which rules to follow. - No central authority needed—though you still trust cryptographic assumptions, code, and infrastructure.

You understand why blockchain makes tyranny auditable: - Everyone has a copy of the history. - Changes are immediately visible. - Can't rewrite the past without redoing enormous computational work. - Minorities can fork and preserve original truth. - The anti-gaslight machine in action.

You understand why value is consensual: - Gold has value because people agree. - Dollars have value because people agree. (and taxes) - Bitcoin has value because people agree. - Value is always social consensus—it always has been in large societies. - Bitcoin just makes this transparent.

You understand why this matters for power: - It shifts control from centralized gatekeepers to distributed networks (though power is not equally distributed in practice). - You can own without intermediaries (if you manage keys properly). - You can coordinate with minimized trust requirements. - You can exit if you disagree (though practical barriers exist). - Power is redistributed differently, not eliminated.

You understand why technology isn't magic: - Computers follow rules (algorithms). - Cryptography is math (one-way functions, signatures, proofs). - Incentives are game theory (selfish actors, aligned interests). - Consensus is social (humans decide, code enforces). - No magic. Just math, engineering, and social coordination.

19.4 You Can Now...

Explain Bitcoin to your grandmother—okay, at least to your mum:

"It's a database that thousands of computers synchronize together." Or just hand them this book.

Critique crypto hype intelligently: - Does this project need a blockchain, or would a normal database work? - Are the incentives aligned properly? - Is this solving a real coordination problem, or just buzzwords? - What are the trade-offs? (Responsibility, complexity, energy, etc.)

Evaluate new projects: - **Ask:** Can this benefit from permissionless participation? Censorship resistance? Transparent history? No central authority? - **If yes:** Maybe a CDN makes sense. - **If no:** Just use a database. Simpler, faster, cheaper.

Participate informed: - Buy, build, regulate, critique, use, or ignore—your choice. - But now you can do it with understanding, not hype. - You know the trade-offs. You know the risks. You know what's possible.

Think clearly about coordination problems: - When do we need trust? When can we avoid it? - When is centralization better? When is decentralization better? - What are the incentives? Who benefits? Who loses? - Is this technology appropriate for this problem?

19.5 The Meta-Insight

Here's the deeper realization:

You learned this from first principles.

We didn't start with "Bitcoin is a decentralized cryptocurrency using Proof-of-Work consensus on a distributed ledger where users send transactions." That sentence would have been gibberish on page 1.

Instead, we started with a light switch. Then we built meaning. Then algorithms. Then protocols. Then cryptography. Then consensus. Then Bitcoin.

Step by step. Layer by layer. Concept by concept.

No jargon handwaving. No "trust me, it's complicated." No shortcuts.

You EARNED this understanding.

And that's important, because now you *intuitively* understand. Not that much of a surface-level anymore. Not by blindly repeating buzzwords. But first-principles understanding.

You can't be fooled that easily by hype. You can think critically. You can evaluate claims.

This is true education.

19.6 From Bit to Bitcoin

Let's close the loop.

We started with a single bit: on or off, 1 or 0—the foundation of all digital information.

From there, we built: - **Meaning:** Humans assign logos to bit patterns (Chapter 2). - **Computation:** Algorithms manipulate bits (Chapter 3). - **Communication:** Protocols coordinate computers (Chapter 4). - **Security:** Cryptography protects information (Chapters 5-7). - **Coordination:** Consensus synchronizes databases (Chapters 9-11). - **Innovation:** Smart contracts, Layer 2s, Zero-Knowledge Proofs (Chapters 13-16). - **Philosophy:** What this means for humanity (Chapter 17). - **Reality check:** The devil is in the details (Chapter 18).

From bit to Bitcoin.

From math to meaning.

From code to consensus.

From technical impossibility to reality.

And now you understand it all.

Not because you memorized definitions, but because you built the understanding from the ground up.

You started with a light switch. You ended with global coordination technology.

That's the journey. That's the achievement. Anything can be learned if broken down enough and with patience. Go learn anything.

Key Insight: You've come so far. From a single bit to understanding global coordination technology. You learned from first principles—no shortcuts, no jargon handwaving. You now understand how strangers coordinate without trust, why blockchain makes tyranny auditable, why value is consensual, why this matters for power, and why technology isn't magic. You can explain Bitcoin clearly, critique hype intelligently, evaluate projects, and participate informed. You EARNED this understanding. From bit to Bitcoin. From math to meaning. From code to consensus. You understand it all now.

One more chapter remains: a personal letter from me to you about why this matters, what I hope you take away, and what comes next. See you there.

20

Chapter 20: Epilogue

The end is here, just to start again.

You made it, congrats. :)

Nineteen chapters. From bit to Bitcoin. From light switches to global coordination technology.

Thank you for trusting me to guide you through this journey.

Now, let me close this book with some final words to give you an even deeper insight on how this technology can and cannot affect our lives.

20.1 Why I Wrote This

I had an interview on a YouTube channel to explain potential use cases of this new technology. There, I realized there was no middle ground: either you stay too vague (trying to hide complexity) and people think you're a scammer, or you use technical words and no one understands you.

I must admit my communication skills are poor when talking about technical things to non-technical audiences. Yet even beyond my limitations, I felt the industry had no proper way of truly explaining things:

The impossible communication problems:

- “**Bitcoin is a digital asset.**” (That’s half the story at best—remember, we interpret the information)
- “**Blockchain networks.**” (What the hell is that? Even computer science students need time to understand a blockchain data structure. How can everyday people hear that word and intuitively grasp it?)
- “**Where should I invest?**” (That question always appears, but again, it’s only half the story. People care about money, not use cases. Even if I tried to answer, how do I properly explain the risk differences between a cryptocurrency coded at the blockchain level versus at the smart contract level?)
- **The contradiction trap:** You say things like “crypto is controlled by no one, like Ethereum,” but then you say “some crypto ON Ethereum are controlled”? You sound contradictory. People think you’re an impostor. (Now you understand how both statements are true simultaneously.)

- **The foundation problem:** People don't know what the internet really is. If they don't know that, how will they understand something as complex as a CDN?
- **The double-ignorance problem:** People—at least in my country, Spain—have little to no knowledge about finance, combined with little knowledge about how to think properly about modern technologies like computers. The result? They have no chance of feeling comfortable thinking about something that combines both. You try to find analogies with mechanics they do understand, but as you've seen, this technology's complexity means your analogy will always be incomplete. Furthermore, because this technology enables genuinely new capabilities, there may simply be no good analogy for them. And when people notice the gaps or contradictions in your explanations, they won't trust you. Again, you sound like an impostor.

I received lots of criticism about this in YouTube comments, and about my poor communication skills for technical topics. That was mainly because my brain was facing all these complexities in real time.

I realized: If we really want people to feel comfortable with “crypto,” we cannot skip the complexity. We must break it down as much as possible and teach it. So we don't sound like scammers no matter what we say.

Therefore, here I am, writing this book.

If you're curious, the video is in Spanish: <https://www.youtube.com/watch?v=j2k52WvLAcg>

20.1.1 The Emotional Barrier

You cannot make people feel comfortable and rational about something new and complex when you trigger their emotions toward something as political and real as money.

Especially in a struggling country like mine, where people fight to make ends meet and Spain's real wages and GDP per capita have been **largely stagnant over the last two decades**. Money is not a joke—especially after so many crises, scammers selling useless courses, everyone trying to waste your money online. If someone tries to explain new things honestly and for free, people distrust you too, because that's exactly how scammers look at first. They gain your trust, then trick you into paying.

I could not figure out another way around this than to improve my communication skills and write this book. The second, as you see, is done.

20.1.2 The Book's Goal

This book's main goal: **teach people to think correctly about these new technologies that shape and influence our lives, from the bit to Bitcoin, so they can know what they are and how to use them safely.**

I tried to abstract away technical jargon as much as possible. When necessary, I provided intuitive explanations first so you can think about each concept as a piece, a tool, with certain features and uses in other systems.

The deeper goal: The ability to think clearly about complex topics. The ability to evaluate claims critically. The ability to see through buzzwords and ask the right questions. I hope you got a feeling for that in this book, and I hope you can extrapolate this to other fields in life.

20.2 The Iron Law of Oligarchy - Why Education Matters

I'll finish with a conclusion from a German sociologist from the last century, Robert Michels, that I think applies powerfully to this new technology.

In his book "*Political Parties: A Sociological Study of the Oligarchical Tendencies of Modern Democracy*" (first published in German in 1911, English translation in 1915), he analyzed how democratic organizations at scale—even those with honest and good intentions—tend to become oligarchies over time. Power concentrates in the hands of a few, and the original ideals of the group are often compromised.

CDNs are nothing other than big organizations at scale. Will they face the same fate Robert Michels analyzed in all human coordination? Will new technologies help us avoid the oligarchical tendencies of human organizations?

Likely not. **But they will make everything more transparent, auditible, and harder to hide from the public eye.** Oligarchies aren't bad on their own—they're bad when they contradict and go against the people they're supposed to represent.

Transparency, speed of information availability, and auditability are powerful tools against these cases. They create a more powerful, more informed public that can quickly align their unavoidable oligarchs with their original goals.

At the very last pages of the second edition of his book, he wrote:

A broader education gives people a greater capacity to exercise oversight and control. We see every day that among the wealthy, however great, the authority of leaders over their followers is never as unrestricted as it is among the poor. In the mass, the poor are powerless and defenseless before their leaders. Their intellectual and cultural inferiority makes it impossible for them to understand where the leader is taking them or to foresee the importance of his actions. Therefore, the great task of social education is to raise the intellectual level of the masses so that, as far as possible, they can counteract the oligarchical tendencies of the working-class movement.

He mainly analyzed the working class socialist movement of his era, but he made clear these patterns remain true across contexts. The general trait: democratic human organizations at scale tend to form oligarchies.

20.2.1 The Oligarchy of Nerds

We, the nerds—the ones who build the code, who can read the code, who can write the code—will become a strong oligarchy if this technology is massively adopted. And because we need lots of nerds to build such a massive codebase, we will have an oligarchy inside the oligarchy.

That is unavoidable. If you want to know more about why, read Robert's book. It was written over a century ago but feels as if it were written today.

We must educate what Robert calls “the masses”—the everyday people—so they can question us and make the system even more resilient to corruption and power abuse. By design, CDNs are very resistant. But design alone isn't enough.

Remember the Ethereum DAO hack that led to a fork? Some critics of the “reverse the hack” option said: *“This is only happening because a big figure in the group, the creator Vitalik Buterin, is positioning himself toward that option. What kind of consensus is this? The consensus of one man?”*

They were sharp in this observation. This is just one of many oligarchic tendencies humans have by nature. Usually the masses (all node runners) don't have as much technical knowledge as the leaders, and they sort of “blindly” trust their opinions and follow them.

Yes, the community decided. But to what extent was this decision influenced by oligarchic tendency? Is this really consensus?

Here is the deepest and most complex real risk of the industry right now: We need more nerds and even more importantly, we need people to better understand what we do.

Otherwise we will become, as Robert Michels studied, an oligarchy of nerds with the potential to go against the interests of the masses.

A small spark of this phenomenon already seemed to happen.

We do not just need you to learn for the technology to work due to technical reasons, but also due to unavoidable human social tendencies.

This is too why I wrote this book.

20.3 Author Permission - This Book Is Free Forever

You can copy it. Share it. Remix it. Improve it. Translate it. Fork it.

Just like Bitcoin itself, this knowledge belongs to everyone.

If you found value in this book, the best thing you can do is share it with people.

Knowledge should be free. Understanding should be accessible.

That's why I wrote this. That's why it's free forever.

We do not thrive when resources are scarce, and knowledge is a resource.

20.4 Why This Matters Beyond Technology

Societies that trust each other generate more wealth. Knowledge is a resource. We do not thrive when resources are scarce.

Those who understand and carefully explore this new technology will have advantages over those who dismiss it entirely. Yet, as revolutionary as it is, fully adopting it overnight would be reckless and impossible.

Remember: When cars were invented, they didn't magically have roads and traffic rules in 10 years. It took generations to build the infrastructure, establish safety standards, and integrate them into daily life for their actual benefits to materialize.

Same goes here.

This technology allows people who don't trust each other—even people who have betrayed each other—to find new ways of coordinating. To say: “Let's try again, but this time we'll use decentralized datasync technologies like CDNs to minimize trust requirements.”

That's the promise. Not perfection. Not utopia. Just new tools for an old problem: how to coordinate when trust is scarce.

20.5 From Bit to Bitcoin

And so we return to where we started—but you're not the same person who opened this book.

You began with a light switch, wondering what a “bit” really meant. Now you understand how that simple on-or-off distinction scales into global coordination technology. You've seen how cryptography creates trust between strangers, how consensus emerges from chaos, and how code can enforce agreements that words alone cannot.

More importantly, you've learned to think clearly about complex systems. Not to accept buzzwords at face value. Not to dismiss what you don't understand. But to ask: *What problem does this actually solve? What are the trade-offs? Who controls what? How can I understand this better breaking it down in smaller concepts?*

This matters beyond crypto. The same mental tools—breaking down complexity, questioning assumptions, seeing through hype—apply everywhere: to AI, to social media algorithms, to any technology that shapes your life while hiding its mechanics.

The oligarchy of nerds is real, and it will grow. But now you're better equipped to question us. To hold us accountable. To participate in decisions about technologies that affect you. That's not nothing—that's exactly what Robert Michels said the masses need.

This knowledge is yours now. Share it. Use it. Build with it.

From you to the future.

Thank you for being part of this journey.

Faithfully, The Author, Charles D. Cheerful

