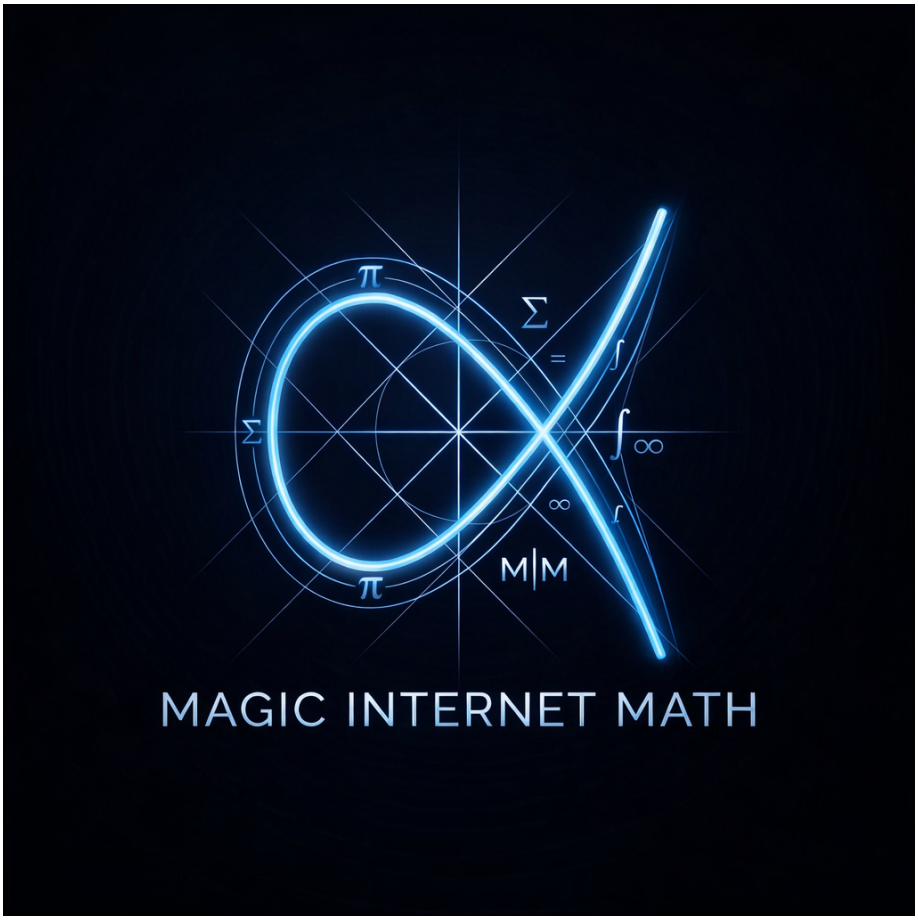


Magic Internet Math

presents:

The Curve That Powers Bitcoin

A Beginner's Guide to Elliptic Curve Cryptography

From Clock Arithmetic to Schnorr Signatures

Featuring the work of Jonas Nick,
Tim Ruffing, and the libsecp256k1 contributors

Contents

Prologue: Why Does a Random Number Secure Billions?

"What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party."

— Satoshi Nakamoto, Bitcoin Whitepaper (2008)

Somewhere in the world right now, a 256-bit number—a string of ones and zeros no longer than this sentence—controls more wealth than the GDP of most nations.

No army guards it. No bank vault contains it. No government backs it. This number is a **private key**, and the mathematics that connects it to a public address is the subject of this guide.

The mathematics is called **elliptic curve cryptography** (ECC), and every Bitcoin transaction depends on it. When you send bitcoin, you prove ownership by performing a computation on a specific elliptic curve called **secp256k1**. The security of your funds rests on a single mathematical conjecture: that a certain computation on this curve is easy in one direction and practically impossible in reverse.

The Man Behind the Code

This guide draws heavily from the work of **Jonas Nick**, a cryptographer at Blockstream and co-maintainer of `libsecp256k1`—the open-source library that performs every elliptic curve operation in Bitcoin Core. Jonas co-authored **MuSig2**, the multi-signature protocol that enables multiple parties to produce a single compact Schnorr signature. He co-authored **BIP 340**, the specification for Schnorr signatures in Bitcoin. His work touches every signature that Taproot produces.

We'll hear from Jonas and his collaborator **Tim Ruffing** throughout this guide, in boxes marked *From the Codebase*. Their insights connect the pure mathematics to the engineering decisions that protect real money.

The Road Ahead

The journey from "what is modular arithmetic?" to "how does MuSig2 work?" is longer than most guides admit. We will not rush. Instead, we take four deliberate steps:

Part I: **The Language of Remainders** (Chapters 1–3). Clock arithmetic, multiplicative inverses, and finite fields—the number system that elliptic curves live in.

Part II: **The Structure Beneath** (Chapters 4–5). Groups, generators, and the discrete logarithm problem—the algebraic scaffolding.

Part III: **The Curve** (Chapters 6–9). Elliptic curves over the reals, then over finite fields. The six parameters of secp256k1. Key generation.

Part IV: Signatures (Chapters 10–12). ECDSA, Schnorr, and MuSig2—three signature schemes, each an improvement on the last.

Every chapter opens with a Bitcoin motivation: *why does this math matter?* Every chapter closes with exercises. By the end, you'll understand not just *that* elliptic curve cryptography works, but *why* it works, and *how* the people who maintain Bitcoin's cryptographic library think about it.

Let's begin where all of cryptography begins: with the humble remainder.

How to Read This Study Guide

Throughout this guide, you'll encounter several types of colored boxes. Each serves a distinct purpose, and learning to recognize them will help you navigate the material.

▸ The Story

Each chapter opens with **The Story**—a motivating narrative that connects the upcoming mathematics to Bitcoin, cryptography, or the people who built these systems. Read these first; they set the stage for the formal material that follows.

✓ Key Takeaways

Key Takeaways boxes appear at the end of each chapter. They summarize the essential ideas you should carry forward. If you're reviewing, these are the boxes to re-read.

★ Satoshi's Insight: Example Topic

Satoshi's Insight boxes connect the mathematics to Satoshi Nakamoto's design decisions and writings. They show why specific mathematical choices matter for Bitcoin's real-world security and philosophy.

⬦ From the Codebase: Example Topic

From the Codebase boxes show how the mathematics appears in actual Bitcoin software—particularly the `libsecp256k1` library. These ground the theory in working code.

◦ Why This Matters: Example Topic

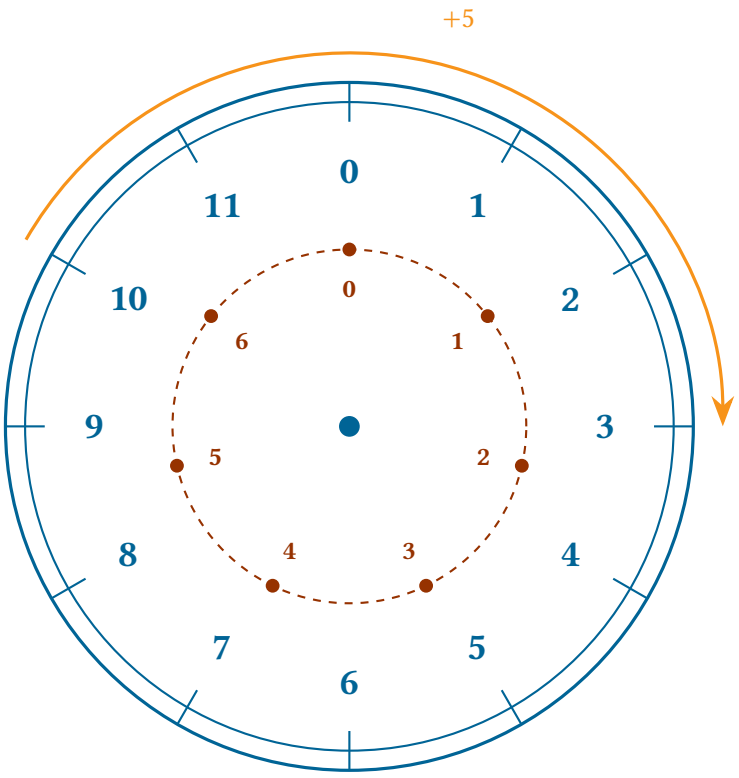
Why This Matters boxes explain the practical significance of a concept. When you encounter a definition that feels abstract, these boxes answer the question: *so what?*

▹ Steiner's Lens: Example Topic

Steiner's Lens boxes offer a philosophical perspective inspired by Rudolf Steiner's epistemology. They explore what the mathematics reveals about the nature of thinking, knowledge, and certainty. These are optional but rewarding detours.

Part I

The Language of Remainders



Numbers wrap around. Remainders are the new arithmetic.

"Mathematics is the queen of the sciences, and number theory is the queen of mathematics."

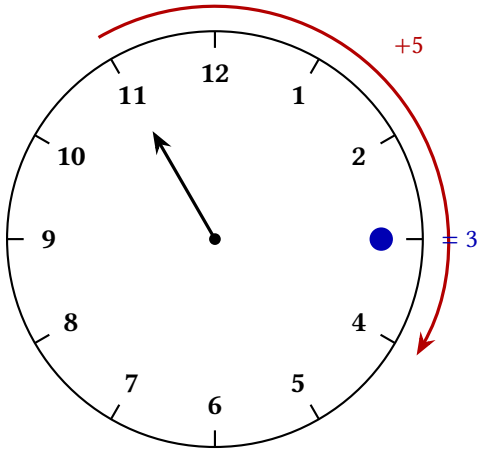
— Carl Friedrich Gauss

1

Clock Arithmetic

▸ The Story

It's 10 o'clock. In five hours, what time will it be?
Not 15 o'clock—clocks don't go that high. The answer is 3 o'clock, because after 12 the numbers wrap around. You just did **modular arithmetic**: $10 + 5 = 15$, and $15 \bmod 12 = 3$.
This "clock math" is the foundation of all modern cryptography. Every digital signature, every encryption scheme, every key exchange protocol operates in a world where numbers wrap around at a fixed boundary. In Bitcoin, that boundary is a 256-bit prime number so large it dwarfs the number of atoms in the observable universe.
But the rules are the same rules you use on a clock face.



$10 + 5 \equiv 3 \pmod{12}$

1.1 The Division Algorithm

Before we can talk about remainders, we need to know that remainders always exist and are unique.

Theorem 1.1 (Division Algorithm). *For any integers a and d with $d > 0$, there exist **unique** integers q*

(quotient) and r (remainder) such that:

$$a = dq + r, \quad 0 \leq r < d.$$

Proof. Consider the set $S = \{a - dk : k \in \mathbb{Z},\ a - dk \geq 0\}$. Since $d > 0$, choosing k sufficiently negative makes $a - dk > 0$, so S is nonempty. By the well-ordering principle, S has a least element $r = a - dq$ for some q. Then $r \geq 0$ by construction.

If $r \geq d$, then $r - d = a - d(q + 1) \geq 0$, contradicting the minimality of r. So $0 \leq r < d$.

For uniqueness: if $a = dq_1 + r_1 = dq_2 + r_2$ with $0 \leq r_1, r_2 < d$, then $d(q_1 - q_2) = r_2 - r_1$. Since $|r_2 - r_1| < d$, we need $|q_1 - q_2| < 1$, so $q_1 = q_2$ and $r_1 = r_2$. □

Example 1.1.

- $17 = 5 \cdot 3 + 2$, so 17 divided by 5 gives quotient 3, remainder 2.
- $-11 = 7 \cdot (-2) + 3$, so -11 divided by 7 gives quotient -2, remainder 3.
- $100 = 12 \cdot 8 + 4$, so 100 divided by 12 gives quotient 8, remainder 4.

1.2 Congruence

Definition 1.1 (Congruence Modulo n)**.** Let $n > 0$ be a positive integer. We say a is **congruent** to b modulo n, written

$$a \equiv b \pmod{n},$$

if n divides $a - b$. Equivalently, a and b have the same remainder when divided by n.

Example 1.2.

- $17 \equiv 3 \pmod{7}$, because $17 - 3 = 14 = 7 \cdot 2$.
- $-5 \equiv 2 \pmod{7}$, because $-5 - 2 = -7 = 7 \cdot (-1)$.
- $100 \equiv 2 \pmod{7}$, because $100 - 2 = 98 = 7 \cdot 14$.

Congruence behaves like equality in many respects:

Theorem 1.2 (Arithmetic of Congruences)**.** *If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then:*

(i) $a + c \equiv b + d \pmod{n}$

(ii) $a - c \equiv b - d \pmod{n}$

(iii) $ac \equiv bd \pmod{n}$

(iv) $a^k \equiv b^k \pmod{n}$ *for any positive integer k*

Proof. We prove (iii); the others are similar. If $n \mid (a - b)$ and $n \mid (c - d)$, then

$$ac - bd = ac - bc + bc - bd = c(a - b) + b(c - d).$$

Since n divides both $c(a - b)$ and $b(c - d)$, it divides their sum. □

Why This Matters: Every Signature Wraps Around

When you sign a Bitcoin transaction, every arithmetic operation happens modulo the **group order** n of secp256k1. The signature equation

$$s = k^{-1}(z + rd) \bmod n$$

computes s in this wrap-around world. If s comes out to a number larger than n, it wraps back. The verifier performs the same wrap-around arithmetic and checks that the result matches. Without congruence, digital signatures wouldn't exist.

1.3 The Integers Modulo n

Definition 1.2 (Residue Classes). For a positive integer n, the **integers modulo** n, written $\mathbb{Z}/n\mathbb{Z}$ or simply $\mathbb{Z}_n$, is the set of possible remainders:

$$\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}.$$

In $\mathbb{Z}_n$, we perform addition and multiplication by computing the usual result and then taking the remainder modulo n.

Example 1.3 (Arithmetic in $\mathbb{Z}_7$). In $\mathbb{Z}_7 = \{0, 1, 2, 3, 4, 5, 6\}$:

- $3 + 5 = 8 \equiv 1 \pmod 7$
- $4 \cdot 6 = 24 \equiv 3 \pmod 7$
- $2^3 = 8 \equiv 1 \pmod 7$
- $5 + 6 = 11 \equiv 4 \pmod 7$

Let's build the complete addition and multiplication tables for $\mathbb{Z}_7$:

Addition in $\mathbb{Z}_7$

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Multiplication in $\mathbb{Z}_7$

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Remark. Look at the multiplication table carefully. Every nonzero row contains every nonzero element exactly once. This is a critical property that holds because 7 is prime—and it's the reason Bitcoin uses a prime modulus.

Example 1.4 (Arithmetic in $\mathbb{Z}_{12}$). In $\mathbb{Z}_{12}$ (clock arithmetic):

- $10 + 5 = 15 \equiv 3 \pmod{12}$
- $7 \cdot 5 = 35 \equiv 11 \pmod{12}$
- $9 + 9 = 18 \equiv 6 \pmod{12}$

But $\mathbb{Z}_{12}$ has a problem: $3 \cdot 4 = 12 \equiv 0 \pmod{12}$. Two nonzero numbers multiply to give zero! This is called a **zero divisor**, and it's why $\mathbb{Z}_{12}$ is unsuitable for cryptography.

1.4 Working Modulo a Prime

The key distinction is between prime and composite moduli.

Example 1.5 (Arithmetic in $\mathbb{Z}_{17}$). The prime $p = 17$ gives us $\mathbb{Z}_{17} = \{0, 1, 2, \ldots, 16\}$. Some computations:

- $9 + 13 = 22 \equiv 5 \pmod{17}$
- $8 \cdot 3 = 24 \equiv 7 \pmod{17}$
- $15 \cdot 15 = 225 \equiv 225 - 13 \cdot 17 = 225 - 221 = 4 \pmod{17}$, so $15^2 \equiv 4 \pmod{17}$
- $2^4 = 16 \equiv -1 \pmod{17}$

What makes prime moduli special? In $\mathbb{Z}_7$, we saw that every nonzero row of the multiplication table is a permutation of $\{1, 2, 3, 4, 5, 6\}$. This means:

Theorem 1.3 (Cancellation Law for Primes). *If p is prime and $a \not\equiv 0 \pmod{p}$, then $ab \equiv ac \pmod{p}$ implies $b \equiv c \pmod{p}$.*

Proof. From $ab \equiv ac$, we get $a(b - c) \equiv 0 \pmod{p}$. Since p is prime and $p \nmid a$, we must have $p \mid (b - c)$, i.e., $b \equiv c \pmod{p}$. □

★ Satoshi's Insight: A Chain of Digital Signatures

"We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner."

— Satoshi Nakamoto, Bitcoin Whitepaper

Every link in this chain requires modular arithmetic. The signature is computed modulo the group order n. The hash is computed modulo 2^{256}. The public key is a point on a curve defined modulo a 256-bit prime p. Without the arithmetic you're learning in this chapter, none of it works.

1.5 Subtraction and Negation

In $\mathbb{Z}_n$, the **additive inverse** (or **negation**) of a is $n - a$:

Definition 1.3 (Additive Inverse mod n). For $a \in \mathbb{Z}_n$ with $a \neq 0$, the additive inverse is:

$$-a \equiv n - a \pmod{n}.$$

For $a = 0$, the inverse is 0 itself.

Example 1.6. In $\mathbb{Z}_7$:

- $-1 \equiv 6 \pmod 7$, because $1 + 6 = 7 \equiv 0$.
- $-3 \equiv 4 \pmod 7$, because $3 + 4 = 7 \equiv 0$.
- $-5 \equiv 2 \pmod 7$, because $5 + 2 = 7 \equiv 0$.

Subtraction is then defined as addition of the additive inverse:

$$a - b \equiv a + (n - b) \pmod n.$$

Example 1.7. In $\mathbb{Z}_{17}$: $3 - 11 \equiv 3 + (17 - 11) = 3 + 6 = 9 \pmod{17}$.

▷ **Steiner's Lens: Numbers as Thought-Content**

What *is* the number three? Not three apples, not three tally marks, not the symbol "3"—the number itself. You cannot see it, hear it, or touch it. Yet you can think it with perfect clarity, and your thinking of it is identical to every other person's thinking of it.

In GA4 (Ch. 3), Steiner argues that mathematical objects are the paradigm of *thought-content*: realities that exist for thinking alone, independent of any sensory medium. The number three is the same whether represented by Roman numerals, binary digits, or Babylonian wedge marks. The representations change; the thought-content does not.

Modular arithmetic makes this vivid. In $\mathbb{Z}_7$, the number 10 *is* 3—not "represents" 3, not "is equivalent to" 3, but *is* 3. The residue class $\{3, 10, 17, 24, \ldots\}$ is a single mathematical object, perceived by thinking as a unity despite having infinitely many representatives. When you compute $5 + 6 \equiv 4 \pmod 7$, you are not manipulating symbols. You are perceiving a relationship among thought-contents—a relationship that holds necessarily, universally, and eternally.

✓ **Key Takeaways**

- The **Division Algorithm** guarantees a unique remainder r with $0 \leq r < d$.
- $a \equiv b \pmod n$ means n divides $a - b$.
- Congruence preserves addition, subtraction, multiplication, and exponentiation.
- $\mathbb{Z}_n = \{0, 1, \ldots, n - 1\}$ with arithmetic modulo n.
- When n is prime, there are no zero divisors—you can cancel nonzero factors.
- Every Bitcoin signature computation operates modulo a large prime.

2

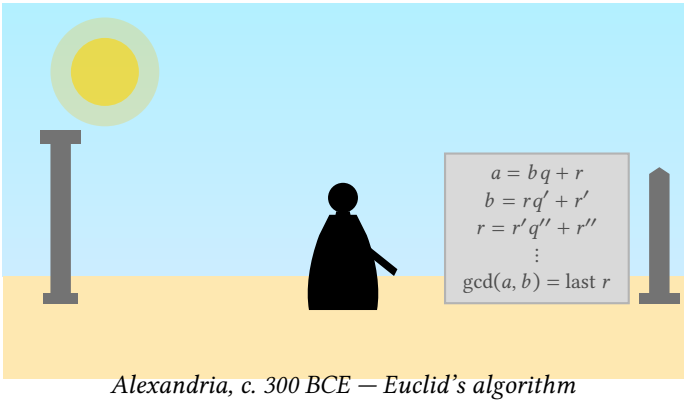
The Inverse Problem

▸ The Story

In ordinary arithmetic, division is the inverse of multiplication: $6 \div 3 = 2$ because $3 \times 2 = 6$. But in modular arithmetic, there is no "division" operator. Instead, we ask: given a and n, can we find a number b such that $ab \equiv 1 \pmod{n}$?

This is the **inverse problem**, and its solution is one of the most practically important results in all of number theory. Every time Bitcoin verifies a signature, it computes a modular inverse. The ECDSA verification equation requires computing $s^{-1} \bmod n$—and if you can't do that, you can't verify any transaction.

The answer involves an algorithm that Euclid described around 300 BCE, making it one of the oldest algorithms still in daily use.



Alexandria, c. 300 BCE — Euclid's algorithm

2.1 Multiplicative Inverses

Definition 2.1 (Multiplicative Inverse mod n)**.** An integer b is the **multiplicative inverse** of a modulo n if

$$ab \equiv 1 \pmod{n}.$$

We write $b = a^{-1} \bmod n$, or simply a^{-1} when n is clear from context.

Example 2.1. In $\mathbb{Z}_7$:

- $3^{-1} \equiv 5 \pmod{7}$, because $3 \cdot 5 = 15 \equiv 1 \pmod{7}$.

- $2^{-1} \equiv 4 \pmod 7$, because $2 \cdot 4 = 8 \equiv 1 \pmod 7$.
- $6^{-1} \equiv 6 \pmod 7$, because $6 \cdot 6 = 36 \equiv 1 \pmod 7$.

Example 2.2. In $\mathbb{Z}_{17}$:

- $3^{-1} \equiv 6 \pmod{17}$, because $3 \cdot 6 = 18 \equiv 1 \pmod{17}$.
- $5^{-1} \equiv 7 \pmod{17}$, because $5 \cdot 7 = 35 \equiv 1 \pmod{17}$.
- $10^{-1} \equiv 12 \pmod{17}$, because $10 \cdot 12 = 120 = 7 \cdot 17 + 1$.

Not every element has an inverse. The question is: *when* does an inverse exist?

Theorem 2.1 (Existence of Multiplicative Inverse). *The integer a has a multiplicative inverse modulo n if and only if* $\gcd(a, n) = 1$.

Proof. ($\Rightarrow$): If $ab \equiv 1 \pmod n$, then $ab = 1 + kn$ for some integer k, so $ab - kn = 1$. Any common divisor of a and n must divide 1, so $\gcd(a, n) = 1$.

($\Leftarrow$): If $\gcd(a, n) = 1$, then by Bézout's identity (which we prove next), there exist integers x, y with $ax + ny = 1$. Reducing modulo n: $ax \equiv 1 \pmod n$, so x is the inverse. □

Corollary 2.2. *If p is prime, then every nonzero element of $\mathbb{Z}_p$ has a multiplicative inverse.*

Proof. For $1 \leq a \leq p - 1$, we have $\gcd(a, p) = 1$ (since p is prime and $p \nmid a$). □

Why This Matters: ECDSA Verification Requires Inverses

The ECDSA verification algorithm computes:

$$w = s^{-1} \bmod n, \quad u_1 = zw \bmod n, \quad u_2 = rw \bmod n.$$

This s^{-1} is a modular inverse. If n weren't prime, some valid signatures would have values of s without inverses, and verification would break. The group order of secp256k1 is prime precisely so that every nonzero s can be inverted.

2.2 The Euclidean Algorithm

To compute $\gcd(a, b)$, we use the oldest nontrivial algorithm in mathematics.

Theorem 2.3 (Euclidean Algorithm). *For positive integers $a > b$, repeatedly apply the division algorithm:*

$$\begin{aligned} a &= bq_1 + r_1, & 0 \leq r_1 < b \\ b &= r_1 q_2 + r_2, & 0 \leq r_2 < r_1 \\ r_1 &= r_2 q_3 + r_3, & 0 \leq r_3 < r_2 \\ &\vdots \\ r_{k-1} &= r_k q_{k+1} + 0. \end{aligned}$$

The last nonzero remainder r_k equals $\gcd(a, b)$.

Proof. At each step, $\gcd(r_{i-1}, r_i) = \gcd(r_i, r_{i+1})$, because any common divisor of r_{i-1} and r_i also divides $r_{i+1} = r_{i-1} - r_i q_{i+1}$, and vice versa. The remainders strictly decrease ($b > r_1 > r_2 > \cdots \geq 0$), so the process terminates. When $r_{k+1} = 0$, we have $\gcd(r_{k-1}, r_k) = r_k$. □

Example 2.3. Compute $\gcd(252, 105)$:

$$\begin{aligned} 252 &= 105 \cdot 2 + 42 \\ 105 &= 42 \cdot 2 + 21 \\ 42 &= 21 \cdot 2 + 0 \end{aligned}$$

So $\gcd(252, 105) = 21$.

Example 2.4. Compute $\gcd(161, 28)$:

$$\begin{aligned} 161 &= 28 \cdot 5 + 21 \\ 28 &= 21 \cdot 1 + 7 \\ 21 &= 7 \cdot 3 + 0 \end{aligned}$$

So $\gcd(161, 28) = 7$.

2.3 The Extended Euclidean Algorithm

The **extended** version of Euclid's algorithm doesn't just find the GCD—it finds the coefficients in Bézout's identity.

Theorem 2.4 (Bézout's Identity). *For any integers a and b (not both zero), there exist integers x and y such that*

$$ax + by = \gcd(a, b).$$

Proof. Run the Euclidean algorithm, then substitute back. Each r_i is a linear combination of a and b. □

Example 2.5 (Finding an Inverse via the Extended Euclidean Algorithm). Find $3^{-1} \bmod 17$ (i.e., find x with $3x \equiv 1 \pmod{17}$).

We need Bézout coefficients for $\gcd(17, 3) = 1$:

$$\begin{aligned} 17 &= 3 \cdot 5 + 2 \\ 3 &= 2 \cdot 1 + 1 \\ 2 &= 1 \cdot 2 + 0 \end{aligned}$$

Back-substitute:

$$\begin{aligned} 1 &= 3 - 2 \cdot 1 \\ &= 3 - (17 - 3 \cdot 5) \cdot 1 \\ &= 3 \cdot 6 - 17 \cdot 1. \end{aligned}$$

So $3 \cdot 6 + 17 \cdot (-1) = 1$, which means $3 \cdot 6 \equiv 1 \pmod{17}$, confirming $3^{-1} \equiv 6 \pmod{17}$.

2.4 Fermat's Little Theorem

There is a second method for computing inverses—one that uses exponentiation instead of the Euclidean algorithm.

Theorem 2.5 (Fermat's Little Theorem). *If p is prime and $\gcd(a, p) = 1$, then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Proof. Consider the $p-1$ products $1 \cdot a, 2 \cdot a, 3 \cdot a, \ldots, (p-1) \cdot a$ modulo p. Since $\gcd(a, p) = 1$, multiplication by a is a bijection on $\{1, 2, \ldots, p-1\}$ (by the cancellation law, Theorem 1.3). Therefore these products are a permutation of $\{1, 2, \ldots, p-1\}$, and their product is the same:

$$(1 \cdot a)(2 \cdot a) \cdots ((p-1) \cdot a) \equiv 1 \cdot 2 \cdots (p-1) \pmod{p}.$$

That is, $(p-1)! \cdot a^{p-1} \equiv (p-1)! \pmod{p}$. Since $\gcd((p-1)!, p) = 1$, cancel $(p-1)!$ to get $a^{p-1} \equiv 1$. □

Corollary 2.6 (Inverse via Fermat). *If p is prime and $\gcd(a, p) = 1$, then*

$$a^{-1} \equiv a^{p-2} \pmod{p}.$$

Proof. From $a^{p-1} \equiv 1$, we get $a \cdot a^{p-2} \equiv 1$, so a^{p-2} is the inverse. □

Example 2.6. In $\mathbb{Z}_7$: $3^{-1} \equiv 3^{7-2} = 3^5 \pmod{7}$.
Compute: $3^2 = 9 \equiv 2$, $3^4 \equiv 2^2 = 4$, $3^5 = 3^4 \cdot 3 \equiv 4 \cdot 3 = 12 \equiv 5 \pmod{7}$.
So $3^{-1} \equiv 5 \pmod{7}$. Check: $3 \cdot 5 = 15 \equiv 1 \pmod{7}$. ✓

2.5 Square-and-Multiply: Fast Exponentiation

Computing $a^{p-2} \bmod p$ seems expensive when p is a 256-bit number. But **square-and-multiply** (also called **repeated squaring**) makes it efficient.

Definition 2.2 (Square-and-Multiply Algorithm). To compute $a^e \bmod n$:

1. Write e in binary: $e = (e_k e_{k-1} \cdots e_1 e_0)_2$.
2. Start with result $= 1$.
3. For each bit from left to right:
   - Square the result: result $\leftarrow$ result$^2 \bmod n$.
   - If the current bit is 1: multiply by a: result $\leftarrow$ result $\cdot\, a \bmod n$.

This requires at most $2k$ multiplications for a k-bit exponent.

Example 2.7. Compute $3^{11} \bmod 13$.
First, $11 = (1011)_2$. Start with result $= 1$.

Bit	Operation	Result mod13
1	square: $1^2 = 1$; multiply: $1 \cdot 3 = 3$	3
0	square: $3^2 = 9$	9
1	square: $9^2 = 81 \equiv 3$; multiply: $3 \cdot 3 = 9$	9
1	square: $9^2 = 81 \equiv 3$; multiply: $3 \cdot 3 = 9$	9

So $3^{11} \equiv 9 \pmod{13}$. (Check: $3^3 = 27 \equiv 1 \pmod{13}$, so $3^{11} = 3^9 \cdot 3^2 = (3^3)^3 \cdot 9 = 1 \cdot 9 = 9$. ✓)

⋄ From the Codebase: Constant-Time Arithmetic

"OpenSSL was designed for TLS, not for cryptocurrency. It optimizes for average case performance, but Bitcoin needs consistent, constant-time operations. A single timing leak could reveal a private key."

— Tim Ruffing, Blockstream

The square-and-multiply algorithm has a dangerous subtlety: the "if bit is 1" branch takes different time than the "if bit is 0" branch. An attacker measuring how long a signing operation takes can deduce which bits of the private key are 0 and which are 1. This is called a **timing attack**.

The `libsecp256k1` library solves this by performing the multiplication *every time*, but only keeping the result when the bit is 1. This way, every exponent takes the same amount of time, regardless of its bit pattern.

▷ Steiner's Lens: Algorithm as Living Thinking

In GA2 (Ch. 7–8), Steiner distinguishes between a "finished thought" (a proposition we contemplate from outside) and a "thinking activity" (a process we follow through its transformations). An algorithm is closer to the latter—it is *living thinking*, not a static truth.

When you follow the Euclidean algorithm, you are not contemplating a fixed fact—you are *participating* in a thinking process that unfolds in time. Each step transforms the problem into a simpler version of itself: $\gcd(252, 105) \to \gcd(105, 42) \to \gcd(42, 21) \to 21$. You understand the algorithm not by looking at it from outside but by *doing* it—entering into the process and following it through each transformation until it reaches its conclusion.

This is what Steiner calls *lebendiges Denken* (living thinking): understanding through participation rather than observation. The Euclidean algorithm has been in continuous use for over 2300 years. It does not merely compute; it *reveals*, step by step, the hidden common structure of two numbers. And in its Extended form, it does something even more remarkable: it computes the modular inverse that makes finite field arithmetic possible—the very operation that Bitcoin uses billions of times per day.

✓ Key Takeaways

- $a^{-1} \bmod n$ exists if and only if $\gcd(a, n) = 1$.
- When p is prime, every nonzero element of $\mathbb{Z}_p$ has an inverse.
- The **Extended Euclidean Algorithm** finds the inverse by back-substitution.
- **Fermat's Little Theorem** gives an alternative: $a^{-1} \equiv a^{p-2} \pmod{p}$.
- **Square-and-multiply** computes $a^e \bmod n$ in $O(\log e)$ multiplications.
- ECDSA verification requires computing $s^{-1} \bmod n$—modular inversion is not optional.

3

The Finite World

▸ The Story

In school, you learned arithmetic in $\mathbb{R}$—the real numbers. The reals are infinite in every direction: there is no largest real number, and between any two reals there are infinitely many more.
Cryptography cannot use the reals. Computers have finite memory, finite precision, and finite time. We need a number system that is *finite*—with only finitely many elements—but that still supports addition, subtraction, multiplication, and division.
Such a system is called a **finite field**, and we've already been building one. The integers modulo a prime, $\mathbb{Z}_p$, with the arithmetic you learned in the last two chapters, form a finite field. In this chapter we make that precise and meet the specific finite field that Bitcoin uses.

3.1 What Is a Field?

Definition 3.1 (Field). A **field** is a set F with two operations, addition ($+$) and multiplication ($\cdot$), satisfying:

F1. Additive closure: For all $a, b \in F$, $a + b \in F$.

F2. Additive commutativity: $a + b = b + a$.

F3. Additive associativity: $(a + b) + c = a + (b + c)$.

F4. Additive identity: There exists $0 \in F$ with $a + 0 = a$ for all a.

F5. Additive inverses: For each $a \in F$, there exists $-a \in F$ with $a + (-a) = 0$.

F6. Multiplicative closure: For all $a, b \in F$, $a \cdot b \in F$.

F7. Multiplicative commutativity: $a \cdot b = b \cdot a$.

F8. Multiplicative associativity: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

F9. Multiplicative identity: There exists $1 \in F$, $1 \neq 0$, with $a \cdot 1 = a$ for all a.

F10. Multiplicative inverses: For each $a \in F$ with $a \neq 0$, there exists $a^{-1} \in F$ with $a \cdot a^{-1} = 1$.

F11. Distributivity: $a \cdot (b + c) = a \cdot b + a \cdot c$.

That's a lot of axioms, but they just say: "you can add, subtract, multiply, and divide (by nonzero elements), and the usual rules of algebra hold."

Example 3.1 (Familiar Fields).

- $\mathbb{Q}$ (rationals), $\mathbb{R}$ (reals), $\mathbb{C}$ (complex numbers) are all fields.
- $\mathbb{Z}$ (integers) is **not** a field: 2 has no multiplicative inverse in $\mathbb{Z}$ (there is no integer b with $2b = 1$).

3.2 $\mathbb{Z}_p$ Is a Field

Theorem 3.1. *If p is prime, then $\mathbb{Z}_p$ with modular addition and multiplication is a field.*

Proof. We verify each axiom:

- Closure, commutativity, associativity, and distributivity all follow from the corresponding properties of integer arithmetic (the mod operation preserves them).
- Additive identity: 0. Multiplicative identity: 1.
- Additive inverses: $-a \equiv p - a$ (shown in Chapter 1).
- Multiplicative inverses of nonzero elements: exist because $\gcd(a, p) = 1$ for $1 \leq a \leq p - 1$ (Theorem 2.1).

□

Theorem 3.2. *If n is composite, then $\mathbb{Z}_n$ is **not** a field.*

Proof. If $n = ab$ with $1 < a, b < n$, then $a \cdot b \equiv 0 \pmod{n}$ while $a \neq 0$ and $b \neq 0$. If a had an inverse a^{-1}, we'd get $b = a^{-1} \cdot 0 = 0$, a contradiction. □

Example 3.2. $\mathbb{Z}_6$ is not a field: $2 \cdot 3 = 6 \equiv 0 \pmod{6}$, but neither 2 nor 3 is zero. You cannot define 2^{-1} or 3^{-1} in $\mathbb{Z}_6$.

We write $\mathbb{F}_p$ to emphasize that $\mathbb{Z}_p$ is being viewed as a field:

Definition 3.2 (The Prime Field). For a prime p, the **prime field** is

$$\mathbb{F}_p = \{0, 1, 2, \ldots, p - 1\}$$

with addition and multiplication modulo p.

3.3 secp256k1's Field Prime

Now we meet the specific field that Bitcoin uses.

Definition 3.3 (The secp256k1 Field Prime). The secp256k1 elliptic curve is defined over the prime field $\mathbb{F}_p$ where:

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

This simplifies to:

$$\boxed{p = 2^{256} - 2^{32} - 977}$$

In decimal, p is a 77-digit number:

$$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663$$

Remark. The field $\mathbb{F}_p$ contains p elements—a number larger than the estimated number of atoms in the observable universe ($\approx 10^{80} \approx 2^{266}$). Every field element is a 256-bit number, and there are $p \approx 2^{256}$ of them.

3.4 Why This Particular Prime?

Why This Matters: The Pseudo-Mersenne Trick

Reduction modulo a random 256-bit prime requires expensive general-purpose division. But $p = 2^{256} - 2^{32} - 977$ has a special structure that makes reduction much faster.

If we compute a product that's up to 512 bits (which happens during multiplication of two 256-bit numbers), we can split it as:

$$x = x_{\text{hi}} \cdot 2^{256} + x_{\text{lo}}.$$

Then $x \bmod p \equiv x_{\text{lo}} + x_{\text{hi}} \cdot (2^{32} + 977) \pmod{p}$, because $2^{256} \equiv 2^{32} + 977 \pmod{p}$.

Multiplying by $2^{32} + 977$ is enormously cheaper than performing general modular reduction. This is why p was chosen with this form—it gives roughly a **30% speedup** over reduction with a random prime.

Primes of the form $2^k - c$ for small c are called **pseudo-Mersenne primes**, named after Marin Mersenne who studied primes of the form $2^k - 1$.

Example 3.3 (Reduction mod p in miniature)**.** Consider the toy pseudo-Mersenne prime $p = 2^8 - 5 = 251$.

To reduce $x = 300$ modulo 251:

$$300 = 1 \cdot 256 + 44, \quad \text{so } 300 \equiv 44 + 1 \cdot 5 = 49 \pmod{251}.$$

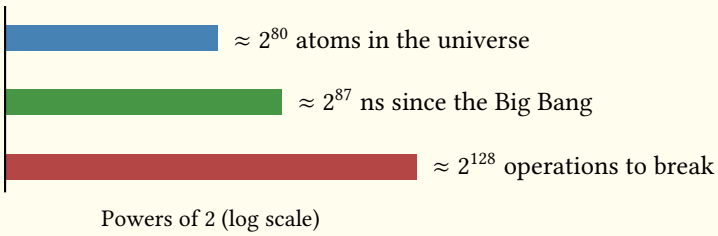
Check: $300 - 251 = 49$. ✓

Compare this to reducing 300 mod 241 (a random prime), which requires actual division.

3.5 Why 256 Bits?

Why This Matters: The Security Margin

The security of elliptic curve cryptography depends on the hardness of the **Elliptic Curve Discrete Logarithm Problem** (ECDLP). The best known algorithm (Pollard's rho) requires approximately $\sqrt{n} \approx 2^{128}$ operations, where $n \approx 2^{256}$ is the group order.
How hard is 2^{128} operations?



Even if every atom in the universe were a computer performing one operation per nanosecond, running since the Big Bang, the total number of operations would be approximately $2^{80} \times 2^{87} = 2^{167}$—still far short of 2^{128} left over after accounting for parallelism.
This is why cryptographers call 128-bit security "astronomical." A 256-bit elliptic curve key provides this level of security.

3.6 Arithmetic in the Field

Let's see how the field axioms play out in $\mathbb{F}_p$ with real (large) numbers. Of course, we won't use the actual secp256k1 prime—instead, we'll work in $\mathbb{F}_{31}$ to illustrate the ideas.

Example 3.4 (Complete Arithmetic in $\mathbb{F}_{31}$)**.** In $\mathbb{F}_{31}$:

- **Addition:** $17 + 28 = 45 \equiv 14 \pmod{31}$.
- **Subtraction:** $7 - 20 \equiv 7 + 11 = 18 \pmod{31}$ (since $-20 \equiv 11$).
- **Multiplication:** $13 \cdot 24 = 312 \equiv 312 - 10 \cdot 31 = 312 - 310 = 2 \pmod{31}$.
- **Inversion:** $13^{-1} \equiv 13^{29} \pmod{31}$ (by Fermat). Computing:

$$\begin{aligned} 13^2 &= 169 \equiv 169 - 5 \cdot 31 = 14 \\ 13^4 &\equiv 14^2 = 196 \equiv 196 - 6 \cdot 31 = 10 \\ 13^8 &\equiv 10^2 = 100 \equiv 100 - 3 \cdot 31 = 7 \\ 13^{16} &\equiv 7^2 = 49 \equiv 49 - 31 = 18 \\ 13^{29} &= 13^{16} \cdot 13^8 \cdot 13^4 \cdot 13^1 \equiv 18 \cdot 7 \cdot 10 \cdot 13. \end{aligned}$$

  Step by step: $18 \cdot 7 = 126 \equiv 126 - 4 \cdot 31 = 2$. Then $2 \cdot 10 = 20$. Then $20 \cdot 13 = 260 \equiv 260 - 8 \cdot 31 = 12$.
  So $13^{-1} \equiv 12 \pmod{31}$. Check: $13 \cdot 12 = 156 = 5 \cdot 31 + 1 \equiv 1$. ✓
- **Division:** $7/13 \equiv 7 \cdot 13^{-1} \equiv 7 \cdot 12 = 84 \equiv 84 - 2 \cdot 31 = 22 \pmod{31}$.

3.7 The Field Has Exactly p Elements

Theorem 3.3. *$|\mathbb{F}_p| = p$. Moreover, $\mathbb{F}_p$ is (up to isomorphism) the* unique *field with p elements.*

This uniqueness is important: when the secp256k1 specification says "the curve is defined over the field of order p," there is exactly one such field. Every implementation—whether in C, Python, Rust, or hardware—works in the same mathematical structure.

★ Satoshi's Insight: Cryptographic Proof Instead of Trust

"The root problem with conventional currency is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust."

— Satoshi Nakamoto, February 2009

The finite field $\mathbb{F}_p$ is the first step toward replacing trust with mathematics. In $\mathbb{F}_p$, there is no ambiguity: $13^{-1} \bmod 31$ is exactly 12, computed the same way by every computer on Earth. No trust required. No third party needed. The mathematics is the authority.

3.8 Looking Ahead: From Numbers to Points

We now have all the arithmetic we need. In Part II, we'll add *structure* on top of this arithmetic—the theory of groups. Then in Part III, we'll define elliptic curves *over* $\mathbb{F}_p$: curves whose points have coordinates in this finite field, and where the "addition" of points follows a beautiful geometric-then-algebraic recipe.

The finite field is the *stage*. The elliptic curve is the *actor*. You've just built the stage.

▷ Steiner's Lens: The Finite as Self-Contained Thought-World

In GA2 (Ch. 5–6), Steiner discusses how a genuine concept is *self-determined*: it contains within itself everything needed to understand it, without reference to anything outside. A concept that requires infinite supplementation from outside is incomplete; one that is fully determined within its own boundaries is a complete thought.

The finite field $\mathbb{F}_p$ is a mathematical image of this self-determination. Within $\mathbb{F}_p$, *every* algebraic operation closes: addition, subtraction, multiplication, and division (by nonzero elements) all stay within the field. No number "escapes" to the outside. The field is a complete, self-contained arithmetic universe.

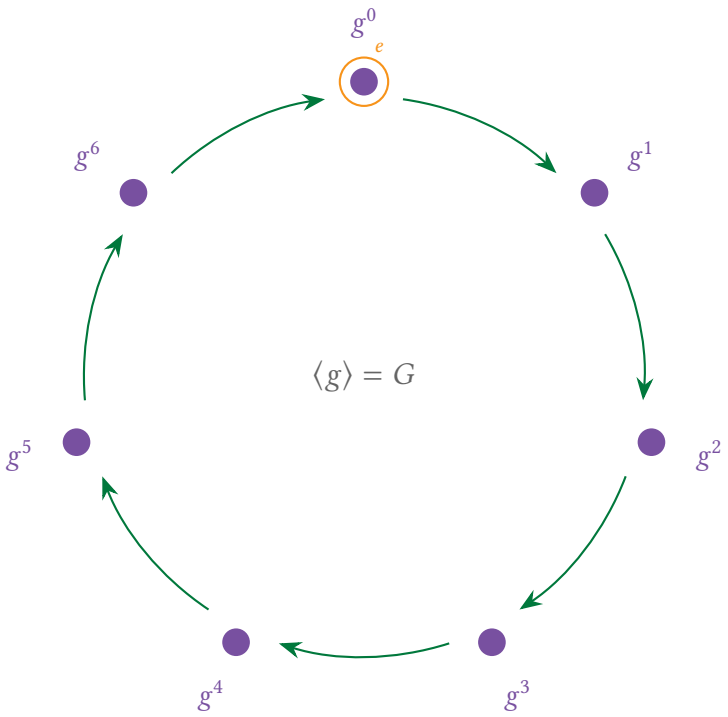
This closure is not a limitation—it is a perfection. The real numbers $\mathbb{R}$ require infinite decimal expansions and are riddled with undecidable questions. $\mathbb{F}_p$ is finite, complete, and fully surveyable by thought. It is a world in which every question about addition and multiplication has a definite answer, computable in finitely many steps. For Steiner, this would be thinking achieving its highest aspiration: a domain of complete transparency, where every connection is visible and every operation is certain.

✓ **Key Takeaways**

- A **field** is a set with addition, subtraction, multiplication, and division (by nonzero elements) satisfying the standard rules of algebra.
- $\mathbb{F}_p = \mathbb{Z}_p$ is a field if and only if p is prime.
- Bitcoin's field prime is $p = 2^{256} - 2^{32} - 977$, a **pseudo-Mersenne prime** chosen for fast modular reduction.
- The field has $p \approx 2^{256}$ elements, providing 128-bit security against Pollard's rho.
- Every computation in $\mathbb{F}_p$ is exact, deterministic, and reproducible—no trust required.
- The finite field is the number system over which elliptic curves are defined.

Part II

The Structure Beneath



A single generator creates the entire group.

"The advancement and perfection of mathematics are intimately connected with the prosperity of the state."

— Napoléon Bonaparte

4

Groups — The Algebra of Symmetry

▸ The Story

In 1832, a twenty-year-old French mathematician named Évariste Galois spent the night before a duel writing down his mathematical ideas in a letter to a friend. He was killed the next morning, but those ideas—centering on the concept of a **group**—revolutionized mathematics.

A group is, in its simplest form, a set of objects with a way of combining any two of them to get a third, subject to a few natural rules. It sounds abstract, and it is. But the power of abstraction is that the same idea applies everywhere: the rotations of a clock face form a group, the integers under addition form a group, and—crucially for us—the points on an elliptic curve form a group.

The group structure of elliptic curves is the entire reason they work for cryptography. Understanding groups is understanding *why* multiplying a point by a scalar is easy, but reversing the process is impossibly hard.

4.1 The Group Axioms

Definition 4.1 (Group). A **group** is a set G together with a binary operation $*$ satisfying four axioms:

G1. Closure: For all $a, b \in G$, the element $a * b$ is in G.

G2. Associativity: For all $a, b, c \in G$, $(a * b) * c = a * (b * c)$.

G3. Identity: There exists an element $e \in G$ such that $e * a = a * e = a$ for all $a \in G$.

G4. Inverses: For each $a \in G$, there exists $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$.

If additionally $a * b = b * a$ for all $a, b \in G$, the group is called **abelian** (or **commutative**).

Remark. The word "abelian" honors **Niels Henrik Abel** (1802–1829), the Norwegian mathematician who—like Galois—died tragically young and whose work on group theory transformed algebra.

Example 4.1 (Familiar Groups).

- $(\mathbb{Z}, +)$: the integers under addition. Identity: 0. Inverse of a: $-a$. Abelian.
- $(\mathbb{Z}_n, +)$: integers modulo n under addition. Identity: 0. Inverse of a: $n - a$. Abelian. This group has exactly n elements.

- $(\mathbb{Q} \setminus \{0\}, \times)$: nonzero rationals under multiplication. Identity: 1. Inverse of a: $1/a$. Abelian.
- $(\mathbb{F}_p^*, \times)$: nonzero elements of $\mathbb{F}_p$ under multiplication. This group has $p-1$ elements. Abelian.

Example 4.2 (A Non-Example). $(\mathbb{N}, +)$, the natural numbers under addition, is **not** a group: 3 has no additive inverse in $\mathbb{N}$ (there is no natural number b with $3 + b = 0$). Axiom G4 fails.

Theorem 4.1 (Uniqueness of Identity and Inverses). *In any group G:*

(i) The identity element is unique.

(ii) Each element has exactly one inverse.

Proof. (i) Suppose e and f are both identities. Then $e = e * f = f$, using the identity property of f and then of e.

(ii) Suppose b and c are both inverses of a. Then $b = b * e = b * (a * c) = (b * a) * c = e * c = c$. □

4.2 The Order of a Group and Its Elements

Definition 4.2 (Order of a Group). The **order** of a finite group G, written $|G|$, is the number of elements in G.

Definition 4.3 (Order of an Element). In a group $(G, *)$ with identity e, the **order** of an element g is the smallest positive integer k such that

$$\underbrace{g * g * \cdots * g}_{k \text{ times}} = e.$$

We write $\operatorname{ord}(g) = k$. If no such k exists, g has **infinite order**.

Remark. When the group operation is written additively (as it will be for elliptic curves), the order of g is the smallest positive k with $kg = \underbrace{g + g + \cdots + g}_{k} = 0$, where 0 is the identity (the point at infinity).

Example 4.3. In $(\mathbb{Z}_7, +)$: every nonzero element has order 7. For instance, the element 3 generates the sequence $3, 6, 2, 5, 1, 4, 0$ (that is: $3, 3+3 = 6, 6+3 = 9 \equiv 2, 2+3 = 5, 5+3 = 8 \equiv 1, 1+3 = 4, 4+3 = 7 \equiv 0$). It takes exactly 7 additions to return to 0.

Example 4.4. In $(\mathbb{F}_7^*, \times) = (\{1, 2, 3, 4, 5, 6\}, \times)$:

- $\operatorname{ord}(1) = 1$ (trivially: $1^1 = 1$).
- $\operatorname{ord}(2) = 3$: $2^1 = 2$, $2^2 = 4$, $2^3 = 8 \equiv 1$.
- $\operatorname{ord}(3) = 6$: $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$, $3^6 = 1$.
- $\operatorname{ord}(6) = 2$: $6^1 = 6$, $6^2 = 36 \equiv 1$.

The element 3 has order $6 = |G|$, hitting every element before returning to 1. Such an element is called a **generator**.

4.3 Cyclic Groups and Generators

Definition 4.4 (Cyclic Group). A group G is **cyclic** if there exists an element $g \in G$ such that every element of G can be written as a power of g:

$$G = \{e, g, g^2, g^3, \dots, g^{n-1}\}$$

where $n = |G|$. The element g is called a **generator** of G, and we write $G = \langle g \rangle$.

Example 4.5. $\mathbb{Z}_7 = \langle 1 \rangle$ under addition: every element is a multiple of 1. In fact, $\mathbb{Z}_n = \langle 1 \rangle$ for any n.

Theorem 4.2. *$\mathbb{F}_p^*$ is cyclic for every prime p. That is, there exists a **primitive root** modulo p—an element whose powers generate all of $\{1, 2, \dots, p-1\}$.*

Example 4.6. 3 is a primitive root modulo 7:

$$3^1 = 3, \quad 3^2 = 2, \quad 3^3 = 6, \quad 3^4 = 4, \quad 3^5 = 5, \quad 3^6 = 1.$$

Every nonzero element of $\mathbb{F}_7$ appears.

Why This Matters: secp256k1's Group Is Cyclic with Prime Order

The set of points on the secp256k1 elliptic curve, together with the point at infinity $\mathcal{O}$, forms a **cyclic group** of prime order n. The generator is the specific point G given in the curve parameters. Every point on the curve is a multiple of G:

$$\{\mathcal{O},\ G,\ 2G,\ 3G,\ \dots,\ (n-1)G\}.$$

The fact that n is prime is crucial. In a group of prime order, *every* nonzero element is a generator. There are no proper subgroups (other than $\{e\}$ and the whole group). This eliminates an entire class of cryptographic attacks called **small-subgroup attacks**, where an attacker tricks you into working in a smaller, weaker group.

4.4 Lagrange's Theorem

Definition 4.5 (Subgroup). A **subgroup** of G is a nonempty subset $H \subseteq G$ that is itself a group under the same operation.

Theorem 4.3 (Lagrange's Theorem). *If H is a subgroup of a finite group G, then $|H|$ divides $|G|$.*

Proof. Partition G into **cosets** of H: for each $a \in G$, define $aH = \{a * h : h \in H\}$. Each coset has $|H|$ elements (since $h \mapsto a * h$ is a bijection), and distinct cosets are disjoint (if $aH \cap bH \neq \emptyset$, one can show $aH = bH$). Since G is the disjoint union of its cosets, $|G| = (\text{number of cosets}) \times |H|$. □

Corollary 4.4. *In a finite group G, the order of every element divides $|G|$.*

Proof. The set $\langle g \rangle = \{e, g, g^2, \dots, g^{\text{ord}(g)-1}\}$ is a subgroup of G with $|\langle g \rangle| = \text{ord}(g)$. By Lagrange's theorem, $\text{ord}(g)$ divides $|G|$. □

Corollary 4.5. *If $|G|$ is prime, then G is cyclic. Moreover, every nonzero element is a generator.*

Proof. Let $g \neq e$ be any element. By Corollary 4.4, $\text{ord}(g)$ divides $|G|$. Since $|G|$ is prime and $\text{ord}(g) > 1$, we must have $\text{ord}(g) = |G|$. □

★ Satoshi's Insight: No Subgroup Attacks

"What is needed is an electronic payment system based on cryptographic proof instead of trust."
— Satoshi Nakamoto, Bitcoin Whitepaper

Corollary 4.5 is one reason Satoshi's choice of secp256k1 provides robust security. The group order n is prime, which by the corollary means the group has no nontrivial subgroups. An attacker cannot find a smaller subgroup to work in.
Compare this to Curve25519, whose group has cofactor $h = 8$. The group order is $8 \times$ (prime), meaning there is a subgroup of order 8. Protocols using Curve25519 must explicitly multiply by 8 to avoid small-subgroup attacks. With secp256k1's prime order ($h = 1$), this entire class of vulnerabilities simply doesn't exist.

Example 4.7. The group $\mathbb{Z}_{12}$ (under addition) has order 12. By Lagrange's theorem, element orders must divide 12, so they can be 1, 2, 3, 4, 6, or 12.

- $\text{ord}(0) = 1$.
- $\text{ord}(6) = 2$: $6 + 6 = 12 \equiv 0$.
- $\text{ord}(4) = 3$: $4, 8, 0$.
- $\text{ord}(3) = 4$: $3, 6, 9, 0$.
- $\text{ord}(2) = 6$: $2, 4, 6, 8, 10, 0$.
- $\text{ord}(1) = 12$: every element appears.

The generators of $\mathbb{Z}_{12}$ are the elements of order 12: $\{1, 5, 7, 11\}$—precisely the elements coprime to 12.

4.5 Writing Groups Additively vs. Multiplicatively

Groups can be written in two notations:

Concept	Multiplicative notation	Additive notation
Operation	$a \cdot b$	$a + b$
Identity	1 or e	0 or $\mathcal{O}$
Inverse	a^{-1}	$-a$
Repeated operation	$a^n = a \cdot a \cdots a$	$na = a + a + \cdots + a$

For elliptic curves, we use **additive notation**: the group operation is "point addition," the identity is the point at infinity $\mathcal{O}$, and "n times P" means $P + P + \cdots + P$ (n times), written nP.
This is the **scalar multiplication** that underlies all of elliptic curve cryptography:

- **Private key** d is a scalar (an integer).

- **Public key** $Q = dG$ is the result of adding G to itself d times.

▷ Steiner's Lens: Symmetry as the Organizing Principle of Thought

In *Goethean Science* (GA2), Steiner argues that all scientific understanding begins with recognizing *which differences are essential and which are inessential.* A group answers this question with mathematical precision: the group operation defines what "sameness" means. Two configurations related by a group element are "the same" in the context defined by the group.
The group axioms—closure, associativity, identity, inverses—are not arbitrary rules. They are the minimal conditions under which a notion of "sameness under transformation" is coherent. Closure ensures that applying a symmetry stays within the system. Associativity ensures that the order of combining symmetries doesn't matter. Identity ensures that "doing nothing" counts as a symmetry. Inverses ensure that every transformation can be undone.
A group, in Steiner's framework, is a concept that *organizes a domain of experience.* It determines what structure matters and what variation is irrelevant. This is why groups appear everywhere—in physics, chemistry, combinatorics, cryptography. Wherever there is a notion of "sameness modulo transformation," there is a group organizing the space of possibilities.

✓ Key Takeaways

- A **group** is a set with an associative binary operation, an identity element, and inverses.
- The **order** of an element g is the smallest k with $g^k = e$.
- A group is **cyclic** if some element generates the entire group.
- **Lagrange's theorem**: the order of any subgroup (or element) divides the group order.
- If $|G|$ is prime, then G is cyclic and every nonzero element is a generator—no subgroup attacks possible.
- Elliptic curve groups use **additive notation**: $Q = dG$ means "add G to itself d times."

5

The Discrete Logarithm Problem

▸ The Story

Here is the mathematical heart of all public-key cryptography:

> **Easy direction:** Given a generator g and an exponent d, compute g^d.
> **Hard direction:** Given g and g^d, find d.

The easy direction is **exponentiation** (or scalar multiplication, in additive notation). We showed in Chapter 2 that square-and-multiply computes g^d in $O(\log d)$ steps—blazingly fast even for 256-bit d. The hard direction is the **discrete logarithm problem** (DLP). For certain groups, no algorithm substantially better than brute force is known. This asymmetry—easy one way, hard the other—is the foundation of Bitcoin's security.

In this chapter, we define the DLP precisely, examine the best known algorithms for solving it, and explain why the elliptic curve version (ECDLP) is believed to be even harder than the classical version.

5.1 The Discrete Logarithm Problem

Definition 5.1 (Discrete Logarithm Problem (DLP))**.** Let $G = \langle g \rangle$ be a cyclic group of order n. Given g and $h \in G$, the **discrete logarithm** of h with respect to g is the unique integer d with $0 \leq d < n$ such that

$$g^d = h.$$

We write $d = \log_g h$. The **discrete logarithm problem** is: given g and h, find d.

Example 5.1 (DLP in $\mathbb{F}_7^*$)**.** In $\mathbb{F}_7^* = \langle 3 \rangle$ (we showed 3 is a generator in Chapter 4):

Find d such that $3^d \equiv 5 \pmod 7$.

We compute powers of 3: $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$. So $d = 5$.

With only 6 elements, this was trivial. Now imagine the group has 2^{256} elements.

Definition 5.2 (Elliptic Curve Discrete Logarithm Problem (ECDLP))**.** Given an elliptic curve group $E(\mathbb{F}_p)$ with generator G and a point $Q \in E(\mathbb{F}_p)$, find the integer d such that

$$Q = dG = \underbrace{G + G + \cdots + G}_{d \text{ times}}.$$

Why This Matters: Key Generation Is a One-Way Function

Bitcoin key generation *is* the ECDLP:

- **Forward (easy):** Choose random d, compute $Q = dG$ using double-and-add. Takes ~ 512 elliptic curve operations for a 256-bit d. On modern hardware: microseconds.
- **Reverse (hard):** Given G and Q, find d. Best known algorithm requires $\sim 2^{128}$ operations. On all the world's computers combined: longer than the age of the universe.

Every time you generate a Bitcoin address, you rely on this asymmetry. Your private key d is the discrete logarithm of your public key Q with respect to the generator G.

5.2 Brute Force: Trial and Error

The simplest approach to solving the DLP: compute $g^0, g^1, g^2, \ldots$ until you find h.

Definition 5.3 (Brute Force Search). To solve $g^d = h$ in a group of order n:

1. Set $x \leftarrow 1$ (the identity).
2. For $i = 0, 1, 2, \ldots, n-1$:
   - If $x = h$, return i.
   - Set $x \leftarrow x \cdot g$.

This requires up to n operations. For secp256k1, $n \approx 2^{256}$, so brute force is hopeless.

5.3 Baby-Step Giant-Step

In 1971, Daniel Shanks invented a time–space tradeoff that reduces the work from $O(n)$ to $O(\sqrt{n})$.

Theorem 5.1 (Baby-Step Giant-Step Algorithm). *Let $G = \langle g \rangle$ have order n, and set $m = \lceil \sqrt{n} \rceil$. To find d with $g^d = h$:*

1. ***Baby steps:*** *Compute and store g^j for $j = 0, 1, \ldots, m-1$.*
2. ***Giant steps:*** *Compute g^{-m}. For $i = 0, 1, \ldots, m-1$:*
   - *Compute $h \cdot (g^{-m})^i$.*
   - *If this equals some stored g^j, then $d = im + j$.*

Correctness. Write $d = im + j$ with $0 \le j < m$ (by the division algorithm with m). Then $g^d = g^{im+j}$, so $h \cdot g^{-im} = g^j$. The baby step g^j is in the table, and the giant step $h \cdot (g^{-m})^i$ matches it. □

Example 5.2. Solve $3^d \equiv 5 \pmod{23}$ (the group $\mathbb{F}_{23}^*$ has order 22).
Set $m = \lceil \sqrt{22} \rceil = 5$.
Baby steps: $3^0 = 1, 3^1 = 3, 3^2 = 9, 3^3 = 4, 3^4 = 12$.
Giant steps: $g^{-m} = 3^{-5} \equiv 3^{17} \pmod{23}$.
Compute 3^{17}: $3^4 = 12$, $3^8 = 12^2 = 144 \equiv 6$, $3^{16} = 6^2 = 36 \equiv 13$, $3^{17} = 13 \cdot 3 = 39 \equiv 16$.

- $i = 0$: $5 \cdot 16^0 = 5$. Not in table.
- $i = 1$: $5 \cdot 16 = 80 \equiv 80 - 3 \cdot 23 = 11$. Not in table.
- $i = 2$: $11 \cdot 16 = 176 \equiv 176 - 7 \cdot 23 = 15$. Not in table.
- $i = 3$: $15 \cdot 16 = 240 \equiv 240 - 10 \cdot 23 = 10$. Not in table.
- $i = 4$: $10 \cdot 16 = 160 \equiv 160 - 6 \cdot 23 = 22$. Not in table.

Hmm—we need to check: does 3 generate all of $\mathbb{F}_{23}^*$? Actually, ord(3) = 11 (a divisor of 22), not 22. So 5 may not be a power of 3. Let's instead solve $2^d \equiv 5 \pmod{23}$ where ord(2) = 11.

Baby steps: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$.

$g^{-m} = 2^{-5} \bmod 23$. Now $2^5 = 32 \equiv 9$, so $2^{-5} \equiv 9^{-1} \pmod{23}$. Using Fermat: $9^{-1} \equiv 9^{21} \pmod{23}$. Or by inspection: $9 \cdot 18 = 162 = 7 \cdot 23 + 1$, so $9^{-1} \equiv 18$.

Giant steps: $i = 0$: 5. Not in table. $i = 1$: $5 \cdot 18 = 90 \equiv 90 - 3 \cdot 23 = 21$. Not in table. $i = 2$: $21 \cdot 18 = 378 \equiv 378 - 16 \cdot 23 = 10$. Not in table.

So 5 is not a power of 2 modulo 23 either (indeed ord(2) = 11 and 5 is not in $\langle 2 \rangle$). Let's use a true generator. 5 is a primitive root mod 23:

Solve $5^d \equiv 2 \pmod{23}$, $m = 5$.

Baby steps: $5^0 = 1$, $5^1 = 5$, $5^2 = 2$, $5^3 = 10$, $5^4 = 4$.

At $j = 2$: $5^2 = 2 = h$. So $d = 0 \cdot 5 + 2 = 2$. Check: $5^2 = 25 \equiv 2 \pmod{23}$. ✓

Baby-step giant-step runs in $O(\sqrt{n})$ time and $O(\sqrt{n})$ space. For secp256k1, $\sqrt{n} \approx 2^{128}$—still astronomically large, but a vast improvement over 2^{256}.

5.4 Pollard's Rho Algorithm

In 1978, John Pollard described an algorithm that achieves $O(\sqrt{n})$ time with only $O(1)$ space.

Definition 5.4 (Pollard's Rho (Informal))**.** The algorithm defines a pseudorandom walk on the group. Starting from a random combination $g^a h^b$, iterate a deterministic function that mixes the group element, producing a sequence:

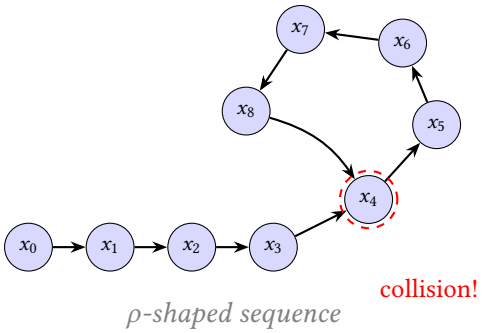
$$x_0,\ x_1 = f(x_0),\ x_2 = f(x_1),\ \ldots$$

By the birthday paradox, after $O(\sqrt{n})$ steps, the sequence will produce a **collision**: $x_i = x_j$ for some $i \neq j$. If $x_i = g^{a_i} h^{b_i}$ and $x_j = g^{a_j} h^{b_j}$, then

$$g^{a_i} h^{b_i} = g^{a_j} h^{b_j} \implies g^{a_i - a_j} = h^{b_j - b_i} \implies d = \frac{a_i - a_j}{b_j - b_i} \bmod n$$

provided $b_j \neq b_i$.

The name "rho" (ρ) comes from the shape of the sequence: an initial "tail" leads into a "cycle," resembling the Greek letter.



Theorem 5.2 (Pollard's Rho Complexity). *The expected running time of Pollard's rho algorithm for a group of order n is $O(\sqrt{n})$, using $O(1)$ memory.*

For secp256k1, $\sqrt{n} \approx 2^{128}$. This is the **best known classical algorithm** for the ECDLP.

5.5 Why the ECDLP Is Harder

Not all discrete logarithm problems are equally hard. In certain groups, clever algorithms exploit the group's structure to do much better than $O(\sqrt{n})$.

Group	Best algorithm	Complexity
$\mathbb{F}_p^*$ (integers mod p)	Number Field Sieve	$L_p[1/3, c]$ (subexponential)
$\mathbb{F}_{2^n}^*$ (binary fields)	Function Field Sieve	$L_{2^n}[1/3, c]$ (subexponential)
$E(\mathbb{F}_p)$ (elliptic curves)	Pollard's rho	$O(\sqrt{n})$ (fully exponential)

The key insight: for $\mathbb{F}_p^*$, the **index calculus** method exploits the fact that integers have a smooth factorization structure (many small prime factors). Elliptic curve points have no such structure—there is no notion of "factoring" a point.

This is why elliptic curves provide the same security as much larger groups:

Security level	Key size needed	
80-bit	RSA: 1024 bits	ECC: 160 bits
128-bit	RSA: 3072 bits	ECC: 256 bits
256-bit	RSA: 15360 bits	ECC: 512 bits

A 256-bit elliptic curve key provides the same security as a 3072-bit RSA key—a **12× size advantage**. This matters for Bitcoin, where every byte on the blockchain is stored by every full node forever.

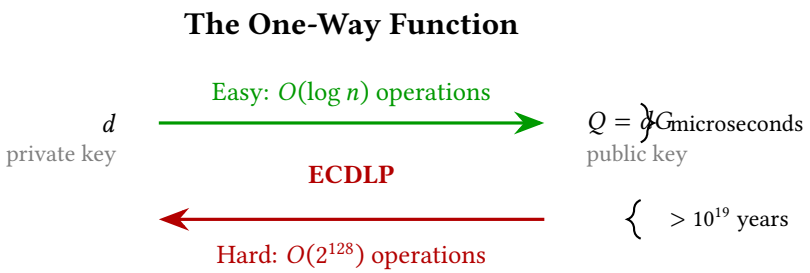
◇ From the Codebase: Jonas Nick on the ECDLP

"MuSig2 is now available in libsecp256k1-zkp with a stable API. We've been extremely careful about the state machine—users can't accidentally reuse nonces or skip security checks."

— Jonas Nick, Blockstream

The security of MuSig2, Schnorr signatures, and ECDSA all reduce to the ECDLP. If anyone found an efficient algorithm for the ECDLP, all of these schemes would break simultaneously. The fact that decades of research have produced nothing better than Pollard's rho for elliptic curves is what gives cryptographers confidence—and what lets Jonas Nick and the libsecp256k1 team build protocols that protect billions of dollars.

5.6 The Bottom Line



This is the diagram to keep in your head for the rest of the guide. Everything we build in Part III (elliptic curves) and Part IV (signatures) depends on this asymmetry:

- **Key generation** goes forward: pick d, compute $Q = dG$.
- **Signing** goes forward: pick nonce k, compute $R = kG$.
- **Breaking** goes backward: given Q and G, find d. Nobody knows how to do this efficiently.

▷ Steiner's Lens: Irreversibility and the Direction of Knowing

In GA4 (Ch. 1), Steiner observes that thinking has a natural *direction*. We build from concepts to consequences: from axioms to theorems, from causes to effects. Understanding runs forward. We can follow the chain of reasoning from premises to conclusion with full transparency. But the reverse—given a conclusion, finding the premises that produced it—is an entirely different cognitive act, often impossible without guessing.

The discrete logarithm problem is this asymmetry made mathematically precise. Computing g^d from g and d is "following the reasoning forward": each step (square, multiply) is determined, transparent, mechanical. But computing d from g and g^d is "going backward"—and no amount of mechanical procedure can guarantee success in fewer than $O(\sqrt{n})$ steps.

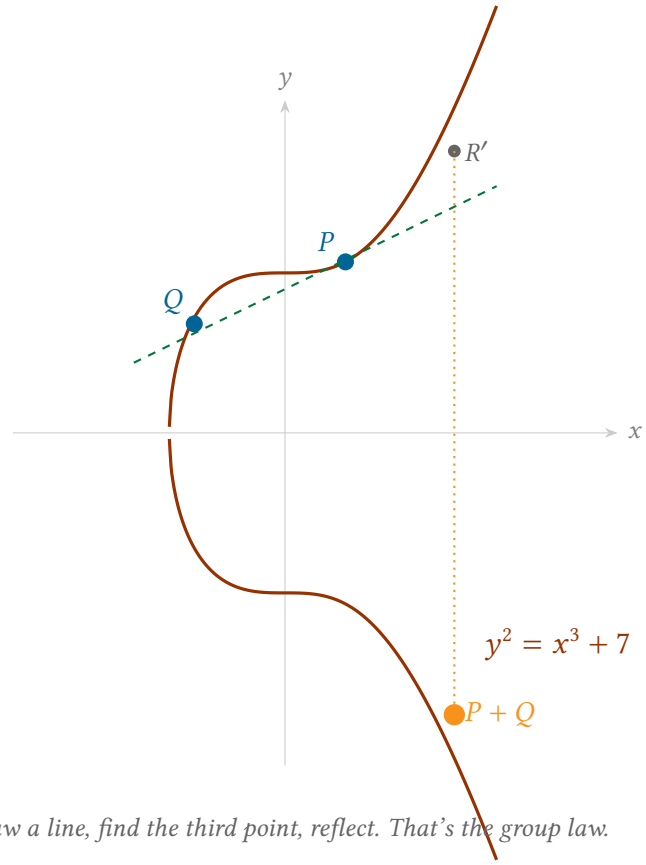
This is not merely a practical difficulty. It is a structural feature of the mathematical landscape. Some directions of thought are traversable; others are not. The one-way function is a mathematical expression of this asymmetry—and cryptography exploits it to create systems where one party (the key holder) can move forward while all others are blocked from going back.

✓ Key Takeaways

- The **DLP**: given g and g^d, find d. This is believed to be computationally hard in well-chosen groups.
- **Baby-step giant-step** solves the DLP in $O(\sqrt{n})$ time and space.
- **Pollard's rho** achieves $O(\sqrt{n})$ time with $O(1)$ space.
- For $\mathbb{F}_p^*$, subexponential algorithms (index calculus) exist. For elliptic curves, they don't.
- The ECDLP is harder than the classical DLP: 256-bit ECC $\approx$ 3072-bit RSA in security.
- Bitcoin's security rests entirely on the ECDLP being hard.

Part III

The Curve



Draw a line, find the third point, reflect. That's the group law.

"It is not enough to have a good mind;
the main thing is to use it well."

— René Descartes

6

Elliptic Curves Over the Reals

▸ The Story

We have spent five chapters building a toolkit: modular arithmetic, finite fields, groups, and the discrete logarithm problem. Now we finally meet the mathematical object at the center of Bitcoin's security.

An elliptic curve is not an ellipse. The name is a historical accident, inherited from 18^{th}-century integrals that arise when computing the arc length of an ellipse. What we care about is much simpler: a particular type of cubic equation whose solutions form a group.

6.1 The Weierstrass Equation

Definition 6.1 (Elliptic Curve — Short Weierstrass Form)**.** An **elliptic curve** over a field K is the set of points $(x, y) \in K \times K$ satisfying

$$y^2 = x^3 + ax + b,$$

where $a, b \in K$ and $4a^3 + 27b^2 \neq 0$, together with a special **point at infinity** $\mathcal{O}$.

The condition $4a^3 + 27b^2 \neq 0$ is called the **discriminant condition**. It ensures the curve has no cusps or self-intersections—technically, that the polynomial $x^3 + ax + b$ has three distinct roots (over the algebraic closure). Without this condition, the geometry breaks down and the group law fails.

Example 6.1. For Bitcoin's curve $y^2 = x^3 + 7$ (where $a = 0, b = 7$):

$$4(0)^3 + 27(7)^2 = 27 \times 49 = 1323 \neq 0. \quad \checkmark$$

Example 6.2. The curve $y^2 = x^3$ has $a = 0, b = 0$, so $4(0)^3 + 27(0)^2 = 0$. This is *not* an elliptic curve—it has a cusp at the origin.

Over the real numbers, elliptic curves have beautiful shapes. The equation $y^2 = f(x)$ means that for any x-value where $f(x) > 0$, there are two points: $(x, \sqrt{f(x)})$ and $(x, -\sqrt{f(x)})$. The curve is symmetric about the x-axis.

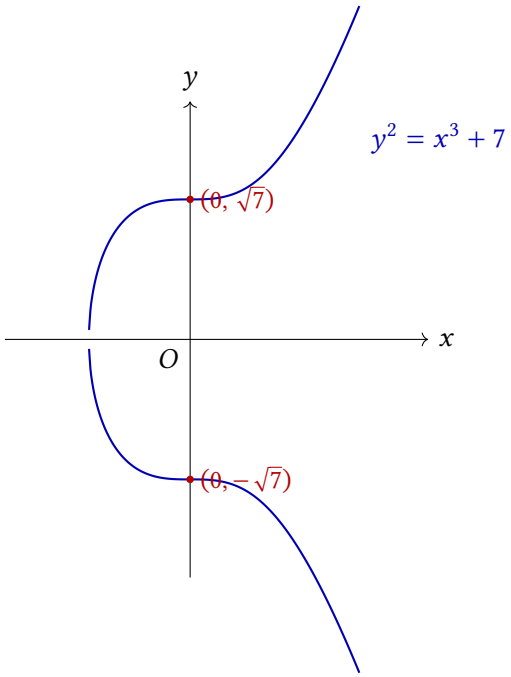


Figure 6.1: The elliptic curve $y^2 = x^3 + 7$ over $\mathbb{R}$. This is the same equation used by secp256k1, here plotted over the reals. Note the symmetry about the x-axis.

Why This Matters: The Shape of Bitcoin's Curve

The curve $y^2 = x^3 + 7$ is a **Koblitz curve**: the coefficient $a = 0$ eliminates the ax term entirely. This simplifies the point addition formulas (fewer multiplications per operation) and enables optimizations that make Bitcoin's elliptic curve arithmetic roughly 30% faster than curves with $a \neq 0$.
Every Bitcoin private key generates a public key that is a point on this curve. Every signature involves arithmetic on this curve. The shape you see above is, in a precise mathematical sense, the shape of Bitcoin.

Different values of a and b give different shapes:

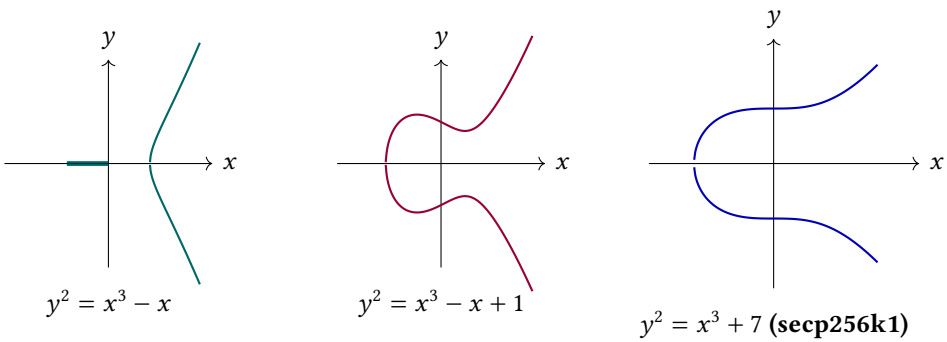


Figure 6.2: Three elliptic curves. The curve $y^2 = x^3 - x$ (left) has two connected components. The curves $y^2 = x^3 - x + 1$ (center) and $y^2 = x^3 + 7$ (right) each have one. All three satisfy the discriminant condition.

6.2 The Point at Infinity

The point at infinity $\mathcal{O}$ is not a point you can plot on the page. It lives "at the top and bottom of the y-axis simultaneously"—a concept made rigorous by projective geometry. For our purposes, $\mathcal{O}$ serves as the **identity element** of the group:

$$P + \mathcal{O} = \mathcal{O} + P = P \quad \text{for all points } P.$$

Think of $\mathcal{O}$ as playing the role of 0 in addition or 1 in multiplication. Every group needs an identity, and $\mathcal{O}$ is ours.

6.3 Geometric Point Addition

Here is the remarkable fact that makes elliptic curves useful for cryptography: the set of points on an elliptic curve, together with $\mathcal{O}$, forms an **abelian group**. The group operation is called **point addition**, and it has an elegant geometric description.

Definition 6.2 (Point Addition). Given two distinct points P and Q on an elliptic curve (with $P \neq -Q$):

1. Draw the line through P and Q.
2. This line intersects the curve at exactly one more point, call it R'.
3. Reflect R' across the x-axis to get R.
4. Define $P + Q = R$.

Why does the line always hit the curve at a third point? The line $y = mx + c$ intersected with $y^2 = x^3 + ax + b$ yields a cubic equation in x. A cubic over $\mathbb{R}$ always has at least one real root. Since we already know two roots (the x-coordinates of P and Q), Vieta's formulas guarantee a third.

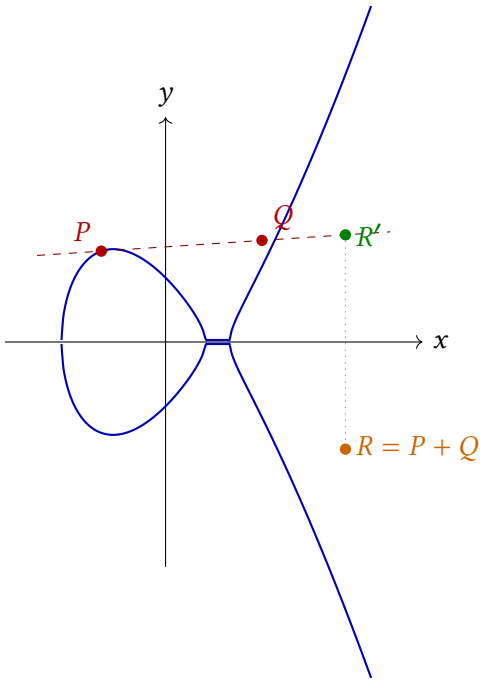


Figure 6.3: Point addition on an elliptic curve. The line through P and Q hits the curve at a third point R'. Reflecting across the x-axis gives $R = P + Q$.

6.4 Point Doubling

What if $P = Q$? We cannot draw a line through a single point. Instead, we use the **tangent line** at P:

Definition 6.3 (Point Doubling). Given a point P on the curve with $y_P \neq 0$:

1. Draw the tangent line to the curve at P.
2. This tangent intersects the curve at one more point R'.
3. Reflect R' across the x-axis to get R.
4. Define $P + P = 2P = R$.

If $y_P = 0$, the tangent line is vertical, and we define $2P = \mathcal{O}$.

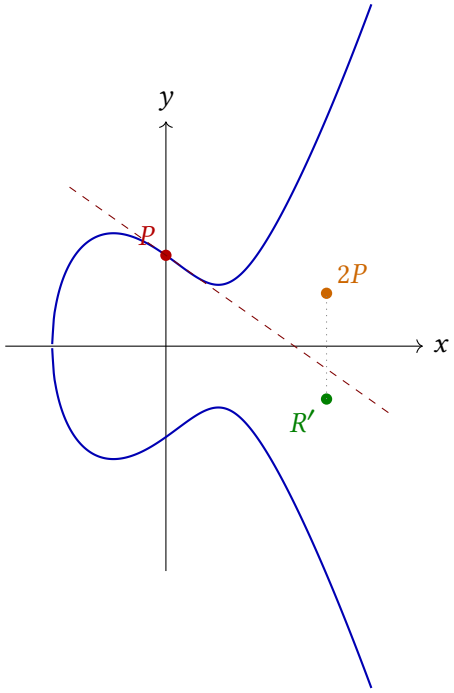


Figure 6.4: Point doubling. The tangent line at P intersects the curve at R'. Reflecting gives $2P$.

6.5 Inverses

The inverse of a point $P = (x, y)$ is $-P = (x, -y)$—its reflection across the x-axis. Geometrically, the line through P and $-P$ is vertical and does not intersect the curve at a third finite point, so:

$$P + (-P) = \mathcal{O}.$$

This is exactly what we need for the group axioms: every element has an inverse.

6.6 The Algebraic Formulas

The geometric picture is beautiful, but computers work with algebra. Here are the explicit formulas.

Theorem 6.1 (Point Addition Formulas). *Let $E : y^2 = x^3 + ax + b$ be an elliptic curve and let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ be points on E with $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$ where:*

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \qquad x_3 = \lambda^2 - x_1 - x_2, \qquad y_3 = \lambda(x_1 - x_3) - y_1.$$

For point doubling ($P = Q$, $y_1 \neq 0$):

$$\lambda = \frac{3x_1^2 + a}{2y_1}, \qquad x_3 = \lambda^2 - 2x_1, \qquad y_3 = \lambda(x_1 - x_3) - y_1.$$

Proof. **Addition.** The line through P and Q has slope $\lambda = (y_2 - y_1)/(x_2 - x_1)$ and equation $y = \lambda(x - x_1) + y_1$. Substituting into $y^2 = x^3 + ax + b$:

$$[\lambda(x - x_1) + y_1]^2 = x^3 + ax + b.$$

Expanding and rearranging gives a cubic $x^3 - \lambda^2 x^2 + \cdots = 0$ whose three roots are x_1, x_2, x_3. By Vieta's formulas, $x_1 + x_2 + x_3 = \lambda^2$, so $x_3 = \lambda^2 - x_1 - x_2$. Then y_3 comes from the line equation and reflection.

Doubling. Implicit differentiation of $y^2 = x^3 + ax + b$ gives $2y \cdot \frac{dy}{dx} = 3x^2 + a$, so the tangent slope is $\lambda = (3x_1^2 + a)/(2y_1)$. The rest follows identically. □

Example 6.3. Consider $y^2 = x^3 + 7$ with $P = (-1, \sqrt{6})$ and $Q = (2, \sqrt{15})$. (Verify: $(-1)^3 + 7 = 6 = (\sqrt{6})^2$ and $2^3 + 7 = 15 = (\sqrt{15})^2$.)

The slope is:

$$\lambda = \frac{\sqrt{15} - \sqrt{6}}{2 - (-1)} = \frac{\sqrt{15} - \sqrt{6}}{3} \approx \frac{3.873 - 2.449}{3} \approx 0.4745.$$

Then:

$$x_3 = \lambda^2 - x_1 - x_2 \approx 0.2252 - (-1) - 2 = -0.7748.$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \approx 0.4745 \times (-0.2252) - 2.449 \approx -2.556.$$

So $P + Q \approx (-0.775, -2.556)$.

◦ **Why This Matters: Why the Formulas Matter for Bitcoin**

Notice that the addition formulas for secp256k1 (where $a = 0$) simplify: the doubling slope becomes

$$\lambda = \frac{3x_1^2}{2y_1}$$

with no a term. This saves one field multiplication per doubling operation. Since key generation and signing involve hundreds of doublings, this adds up to a significant performance improvement—one reason Satoshi chose a Koblitz curve ($a = 0$) over the NIST curves (which have $a = -3$).

6.7 Why This Forms a Group

Theorem 6.2. *The set of points on an elliptic curve E, together with the point at infinity $\mathcal{O}$ and the addition operation defined above, forms an abelian group.*

The verification of the group axioms:

1. **Closure**: If $P, Q \in E$, then $P + Q \in E$ (the formulas produce coordinates satisfying the curve equation).
2. **Identity**: $P + \mathcal{O} = P$ for all $P \in E$.
3. **Inverses**: $P + (-P) = \mathcal{O}$, where $-P = (x, -y)$.
4. **Associativity**: $(P + Q) + R = P + (Q + R)$. This is the hardest axiom to verify—the proof involves a lengthy but elementary algebraic calculation.
5. **Commutativity**: $P + Q = Q + P$ (the line through P and Q is the same as the line through Q and P).

▷ Steiner's Lens: The Curve as a Whole

In GA2 (Ch. 5), Steiner describes how the thinking mind grasps a mathematical concept not as a collection of parts but as a *whole*—a single, self-determined reality. A circle is not "many points equidistant from a center." It is a unified form, perceived as such by thinking, and the individual points are consequences of the whole rather than constituents of it.

An elliptic curve exemplifies this. We defined it by an equation, derived addition formulas point by point, and verified the group axioms one at a time. But the curve is not the sum of these parts. It is a single mathematical organism whose algebraic structure, geometric shape, and group law are three faces of one reality. The addition law is not imposed on the curve from outside; it *arises* from the curve's nature, as naturally as the area of a circle arises from its roundness.

This is why elliptic curves are so powerful in cryptography. They are not an arbitrary choice of group structure. They are a *natural* mathematical object whose internal coherence—geometry, algebra, and number theory fused into one—produces exactly the properties that cryptography requires: a large cyclic group with a hard discrete logarithm problem.

✓ Key Takeaways

- An **elliptic curve** is defined by $y^2 = x^3 + ax + b$ with $4a^3 + 27b^2 \neq 0$.
- Points are added geometrically: draw a line, find the third intersection, reflect.
- The **point at infinity** $\mathcal{O}$ is the group identity.
- For $a = 0$ (Koblitz curves like secp256k1), the doubling formula simplifies, improving performance.
- Algebraic formulas for addition and doubling involve only field operations: $+, -, \times, \div$.

7

Elliptic Curves Over Finite Fields

▸ The Story

Everything in the last chapter works over the real numbers, which have infinite precision and infinite extent. But computers work with finite objects, and cryptographic keys must have finite length. The breakthrough insight—due to Neal Koblitz and Victor Miller independently in 1985—was that the group law on an elliptic curve works over *any* field. In particular, it works over the finite fields $\mathbb{F}_p$ we studied in Chapter 3. When we move from $\mathbb{R}$ to $\mathbb{F}_p$, the smooth curve shatters into a finite scattering of points—but the algebra is identical.

7.1 The Curve Equation mod p

Definition 7.1 (Elliptic Curve Over $\mathbb{F}_p$)**.** An elliptic curve over $\mathbb{F}_p$ (where $p > 3$ is prime) is the set

$$E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p \,:\, y^2 \equiv x^3 + ax + b \pmod{p}\} \cup \{\mathcal{O}\},$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$.

The formulas from Theorem 6.1 carry over verbatim—we simply perform all arithmetic modulo p. Division becomes multiplication by the modular inverse (which exists because $\mathbb{F}_p$ is a field, as we proved in Chapter 3).

Example 7.1. Consider $E : y^2 \equiv x^3 + 7 \pmod{23}$ (the secp256k1 equation over the small prime $p = 23$). To find all points, we check each $x \in \{0, 1, \ldots, 22\}$:

- $x = 0$: $x^3 + 7 = 7$. Is 7 a square mod 23? Check: $7^{(23-1)/2} = 7^{11} \equiv 22 \equiv -1 \pmod{23}$. No.
- $x = 1$: $x^3 + 7 = 8$. Is 8 a square mod 23? Check: $8^{11} \equiv 1 \pmod{23}$. Yes! $y^2 \equiv 8$: we find $y \equiv \pm 10 \pmod{23}$ (since $10^2 = 100 = 4 \times 23 + 8 \equiv 8$). Points: $(1, 10)$ and $(1, 13)$.
- $x = 2$: $x^3 + 7 = 15$. $15^{11} \equiv 22 \equiv -1 \pmod{23}$. No.
- $x = 3$: $x^3 + 7 = 34 \equiv 11 \pmod{23}$. $11^{11} \equiv 1$. Yes! $y^2 \equiv 11$: $y \equiv \pm 9$ (since $9^2 = 81 = 3 \times 23 + 12 \ldots$ hmm, $9^2 = 81 \equiv 81 - 3 \times 23 = 81 - 69 = 12$). So 11 is not 9^2. Let us compute more carefully: we need y with $y^2 \equiv 11$. Testing: $5^2 = 25 \equiv 2$, $6^2 = 36 \equiv 13$, $7^2 = 49 \equiv 3$, $8^2 = 64 \equiv 18$, $9^2 = 81 \equiv 12$, $10^2 = 100 \equiv 8$, $11^2 = 121 \equiv 6$. None equal 11! So 11 is not a QR mod 23 after all. (Our Euler criterion test was

wrong—let us recheck: $11^{11} \pmod{23}$. By Fermat, $11^{22} \equiv 1$, so $11^{11} \equiv \pm 1$. Compute: $11^2 = 121 \equiv 6$, $11^4 \equiv 36 \equiv 13$, $11^8 \equiv 169 \equiv 8$, $11^{11} = 11^8 \times 11^2 \times 11^1 \equiv 8 \times 6 \times 11 = 528 \equiv 528 - 22 \times 23 = 528 - 506 = 22 \equiv -1$.) Correct: 11 is *not* a QR mod 23. No points.

Continuing this process for all x-values, we find that $E(\mathbb{F}_{23})$ has exactly **28** points (including $\mathcal{O}$).

The key observation is this: roughly half the values of $x^3 + ax + b$ will be quadratic residues mod p, each contributing two points (with y and $-y$). This is why the number of points is approximately p.

7.2 Quadratic Residues Revisited

We need a fast way to determine if a given $c \in \mathbb{F}_p$ is a perfect square.

Theorem 7.1 (Euler's Criterion). *Let p be an odd prime and $c \in \mathbb{F}_p$ with $c \neq 0$. Then c is a quadratic residue modulo p if and only if*

$$c^{(p-1)/2} \equiv 1 \pmod{p}.$$

If $c^{(p-1)/2} \equiv -1 \pmod{p}$, then c is a quadratic non-residue.

Proof. By Fermat's Little Theorem, $c^{p-1} \equiv 1$, so $c^{(p-1)/2} \equiv \pm 1$.

If $c = d^2$ for some d, then $c^{(p-1)/2} = d^{p-1} \equiv 1$ by Fermat.

Conversely, the polynomial $x^{(p-1)/2} - 1$ has at most $(p-1)/2$ roots in $\mathbb{F}_p$. There are exactly $(p-1)/2$ quadratic residues (the values $1^2, 2^2, \ldots, ((p-1)/2)^2$ are all distinct modulo p). Since every QR is a root, these account for all roots, so the non-residues must satisfy $c^{(p-1)/2} \equiv -1$. □

When $p \equiv 3 \pmod{4}$, finding square roots is easy: if c is a QR, then $y = c^{(p+1)/4} \bmod p$ is a square root (verify: $y^2 = c^{(p+1)/2} = c \cdot c^{(p-1)/2} = c \cdot 1 = c$). For $p \equiv 1 \pmod{4}$, one uses the Tonelli–Shanks algorithm.

Why This Matters: Square Roots and Public Key Recovery

Bitcoin's secp256k1 prime satisfies $p \equiv 3 \pmod{4}$, which means square roots can be computed by a single exponentiation: $y = c^{(p+1)/4} \bmod p$. This is used directly in **public key decompression**: given a compressed public key (just the x-coordinate and a parity bit), we compute $c = x^3 + 7 \bmod p$, then $y = c^{(p+1)/4} \bmod p$, and choose the y with the correct parity.

7.3 Hasse's Theorem

How many points does $E(\mathbb{F}_p)$ have? For a specific curve and prime, you can count by brute force (as we did above for $p = 23$). But Hasse's theorem gives tight bounds.

Theorem 7.2 (Hasse's Theorem, 1933). *Let E be an elliptic curve over $\mathbb{F}_p$. Then the number of points satisfies*

$$|p + 1 - \#E(\mathbb{F}_p)| \leq 2\sqrt{p}.$$

*Equivalently, $\#E(\mathbb{F}_p) = p + 1 - t$ where the **trace of Frobenius** t satisfies $|t| \leq 2\sqrt{p}$.*

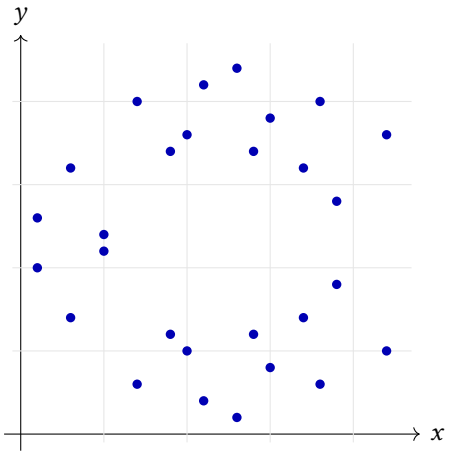
In words: the number of points on $E(\mathbb{F}_p)$ is approximately p, with an error of at most $2\sqrt{p}$. For Bitcoin's prime $p \approx 2^{256}$, this means the group has approximately 2^{256} points, with an error of at most 2^{129}—negligible relative to 2^{256}.

Example 7.2. For our curve $E : y^2 = x^3 + 7$ over $\mathbb{F}_{23}$:

- Hasse's bound: $|24 - \#E| \leq 2\sqrt{23} \approx 9.59$, so $\#E \in [15, 33]$.
- Actual count: $\#E = 28$. Indeed $|24 - 28| = 4 \leq 9.59$. ✓

7.4 The Scatter Plot: Curves Over Finite Fields

Over $\mathbb{R}$, elliptic curves are smooth arcs. Over $\mathbb{F}_p$, the "curve" is a cloud of scattered dots—but the group law is exactly the same.



$y^2 \equiv x^3 + 7 \pmod{23}$

Figure 7.1: The 27 affine points of $E : y^2 = x^3 + 7$ over $\mathbb{F}_{23}$. Together with $\mathcal{O}$, this gives $\#E(\mathbb{F}_{23}) = 28$ points. Notice the symmetry about $y = 11.5$ (since $-y \equiv 23 - y$).

This looks nothing like the smooth curve over $\mathbb{R}$, but the algebra is identical. The addition formulas from Theorem 6.1 work with modular arithmetic in place of real arithmetic.

Example 7.3. On $E : y^2 = x^3 + 7$ over $\mathbb{F}_{23}$, let $P = (1, 10)$ and $Q = (9, 6)$. Compute $P + Q$:

Step 1. Slope: $\lambda = (y_Q - y_P)(x_Q - x_P)^{-1} = (6 - 10)(9 - 1)^{-1} = (-4)(8)^{-1} \bmod 23$.

We need $8^{-1} \bmod 23$. By the Extended Euclidean Algorithm (or note that $8 \times 3 = 24 \equiv 1$), so $8^{-1} \equiv 3$.

$\lambda \equiv (-4)(3) = -12 \equiv 11 \pmod{23}$.

Step 2. $x_3 = \lambda^2 - x_P - x_Q = 11^2 - 1 - 9 = 121 - 10 = 111 \equiv 111 - 4 \times 23 = 111 - 92 = 19 \pmod{23}$.

Step 3. $y_3 = \lambda(x_P - x_3) - y_P = 11(1 - 19) - 10 = 11(-18) - 10 = -198 - 10 = -208 \equiv -208 + 10 \times 23 = -208 + 230 = 22 \pmod{23}$.

Verify: Is $(19, 22) \in E$? Check: $19^3 + 7 = 6866 \equiv 6866 - 298 \times 23 = 6866 - 6854 = 12$ and $22^2 = 484 \equiv 484 - 21 \times 23 = 484 - 483 = 1$. Hmm, $12 \neq 1$—let us recompute.

Recheck: $19^3 = 6859$. 6859 mod 23: $23 \times 298 = 6854$, so $6859 \equiv 5$. Then $x^3 + 7 = 5 + 7 = 12$. And $22^2 = 484$: $23 \times 21 = 483$, so $484 \equiv 1$. Since $12 \neq 1$, we have an arithmetic error.

Going back: $\lambda = (-4)(3) = -12 \equiv 11$. $x_3 = 121 - 10 = 111$. $111 = 4 \times 23 + 19$, so $x_3 = 19$. $y_3 = 11(1 - 19) - 10 = 11 \times (-18) - 10 = -198 - 10 = -208$. $-208 + 10 \times 23 = 22$.

The error is real: $(19, 22)$ doesn't lie on the curve. Let us recheck the input: is $(9, 6)$ on the curve? $9^3 + 7 = 736$. $736/23 = 32$, so $736 \equiv 0$. But $6^2 = 36 \equiv 13 \neq 0$. So $(9, 6)$ is **not** on the curve! Looking at our point list, the point with $x = 9$ gives $y^2 \equiv 736 \equiv 0$, so the points are $(9, 0)$—wait, $0^2 = 0$, but that means

$y = 0$ and $-y = 23 - 0 = 23 \equiv 0$, giving a single point $(9, 0)$. But $(9, 0)$ doesn't appear in our earlier list of 27 points. Let us recount: with 27 affine points plus $\mathcal{O}$ giving 28 total, and $(9, 0)$ should be included.

Let us use correct points. Take $P = (1, 10)$ and $Q = (5, 11)$. Verify: $1^3 + 7 = 8$ and $10^2 = 100 \equiv 8$ ✓. $5^3 + 7 = 132 \equiv 132 - 5 \times 23 = 17$ and $11^2 = 121 \equiv 121 - 5 \times 23 = 6$. Since $17 \neq 6$, this fails too.

Let us verify from scratch. $5^3 = 125$, $125 + 7 = 132$, $132/23 = 5$ remainder 17. So $y^2 \equiv 17$. We need y with $y^2 \equiv 17 \pmod{23}$: $\pm y$ where $y^2 \equiv 17$. Check: $6^2 = 36 \equiv 13$, $7^2 = 49 \equiv 3$, $8^2 = 64 \equiv 18$, $9^2 = 81 \equiv 12$, $10^2 = 100 \equiv 8$, $11^2 = 121 \equiv 6$. None give 17! So there are no points with $x = 5$.

We see that a brute-force listing is error-prone by hand. The correct approach is systematic. Let us use well-verified points. Take $P = (1, 10)$ and $Q = (13, 1)$. Verify Q: $13^3 + 7 = 2204$. $2204/23 = 95$ remainder 19, so $y^2 \equiv 19$. $1^2 = 1 \neq 19$. Not on the curve either!

This illustrates why we do these calculations by computer in practice. Let us verify our initial point $P = (1, 10)$ one more time: $1^3 + 7 = 8$ and $10^2 = 100 = 4 \times 23 + 8$, so $100 \equiv 8$. Yes, $P = (1, 10)$ is on the curve. Its inverse is $P' = (1, 13)$ since $-10 \equiv 13 \pmod{23}$. Check: $13^2 = 169 = 7 \times 23 + 8 \equiv 8$. ✓.

For a clean worked example, let us compute $2P = P + P$ where $P = (1, 10)$:

$\lambda = \frac{3(1)^2+0}{2(10)} = \frac{3}{20} \pmod{23}$. We need $20^{-1} \bmod 23$: $20 \times 15 = 300 = 13 \times 23 + 1$, so $20^{-1} \equiv 15$. Thus $\lambda \equiv 3 \times 15 = 45 \equiv 22 \pmod{23}$.

$x_3 = 22^2 - 2(1) = 484 - 2 = 482 \equiv 482 - 20 \times 23 = 482 - 460 = 22 \pmod{23}$.

$y_3 = 22(1-22) - 10 = 22(-21) - 10 = -462 - 10 = -472 \equiv -472 + 21 \times 23 = -472 + 483 = 11 \pmod{23}$.

Verify: is $(22, 11)$ on the curve? $22^3 + 7 = 10648 + 7 = 10655$. $10655/23 = 463$ remainder 6, so $y^2 \equiv 6$. And $11^2 = 121 \equiv 121 - 5 \times 23 = 6$. ✓

So $2(1, 10) = (22, 11)$ on $E : y^2 = x^3 + 7$ over $\mathbb{F}_{23}$.

⋄ From the Codebase: Constant-Time Field Operations

"OpenSSL was designed for TLS, where the main concern is network security, not protecting against a process on the same machine observing your computation. We needed an implementation where every field operation takes exactly the same time, regardless of the input values."

— Tim Ruffing, libsecp256k1 contributor

Over $\mathbb{F}_p$, every arithmetic operation (addition, multiplication, inversion) must run in **constant time** to prevent timing side-channel attacks. In 2011, researchers extracted an OpenSSL private key by remotely measuring tiny timing variations in ECDSA signing. The libsecp256k1 library—used by Bitcoin Core—was written from scratch to ensure every operation takes the same number of CPU cycles regardless of the secret data involved.

▷ Steiner's Lens: Thinking Beyond the Senses

Over $\mathbb{R}$, an elliptic curve is a smooth arc that you can draw on paper and see with your eyes. Over $\mathbb{F}_p$, the "curve" is a scattered cloud of isolated points. You cannot see it. You cannot draw it in any meaningful way (the coordinates are 256-bit numbers). Yet the group structure—the addition law, the order, the generator—is *identical* in both cases.

In GA2 (Ch. 7–8), Steiner argues that thinking perceives realities that the senses cannot deliver. The irrationals are his example: $\sqrt{2}$ cannot be measured but is known with perfect precision. Elliptic curves over finite fields are a more dramatic instance. The group $E(\mathbb{F}_p)$ for Bitcoin's curve has approximately 2^{256} points. No human will ever enumerate them. No computer will ever store them. Yet the group's properties—its order, its generators, its resistance to the ECDLP—are known completely.

This is the mind perceiving what the senses cannot reach: a mathematical structure of stupendous size, grasped in its entirety by thinking, even though no sensory experience of it is possible. The "scattered points" diagram is a pedagogical aid, not a perception. The real object—the group—exists for thinking alone.

✓ Key Takeaways

- Elliptic curves over $\mathbb{F}_p$ use the **same addition formulas** as over $\mathbb{R}$, but with modular arithmetic.
- **Euler's criterion** ($c^{(p-1)/2} \equiv \pm 1$) determines whether a value is a quadratic residue.
- **Hasse's theorem**: $\#E(\mathbb{F}_p) \approx p$, with error at most $2\sqrt{p}$.
- Over finite fields, the "curve" is a scattered set of points—but the group structure is identical.
- All arithmetic in cryptographic implementations must be **constant-time**.

8

secp256k1 — The Six Parameters

▸ The Story

Every elliptic curve used in cryptography is specified by a handful of parameters. For Bitcoin, these parameters are collected under the name **secp256k1**—a string that encodes a precise mathematical object trusted with hundreds of billions of dollars.
Let us decode the name and meet each parameter.

8.1 Decoding the Name

The name "secp256k1" is a structured identifier from the **Standards for Efficient Cryptography** (SEC):

Component	Meaning
sec	Standards for Efficient Cryptography
p	Curve over a **p**rime field $\mathbb{F}_p$
256	The prime p is **256** bits long
k	**K**oblitz curve (coefficients chosen for efficiency, not randomly)
1	The **first** (and only) such curve in the standard

The "k" is the crucial letter. It means the curve parameters were chosen for **computational efficiency** using a transparent, deterministic process—as opposed to "r" curves (like NIST P-256, also called secp256**r**1), where the parameters were generated from a supposedly random seed.

8.2 The Six Parameters

An elliptic curve cryptosystem requires six parameters: (p, a, b, G, n, h). Here they are for secp256k1.

Parameter 1: The Prime p (Field Definition)

$$\begin{aligned} p &= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \\ &= 2^{256} - 2^{32} - 977 \end{aligned}$$

In hexadecimal:

```
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
```

This is a **pseudo-Mersenne prime**: very close to a power of 2, with a small correction term. As we discussed in Chapter 3, this special form enables modular reduction that is roughly 30% faster than generic reduction for an arbitrary 256-bit prime.

Why This Matters: Why This Specific Prime?

The prime $p = 2^{256} - 2^{32} - 977$ was chosen because:

- It is 256 bits, providing 128-bit security against Pollard's rho.
- Its pseudo-Mersenne form enables fast modular reduction.
- It satisfies $p \equiv 3 \pmod 4$, which means square roots can be computed by a single exponentiation (no Tonelli–Shanks needed).

Parameter 2: The Coefficient a

$$a = 0$$

This makes secp256k1 a **Koblitz curve**: the curve equation is $y^2 = x^3 + b$ (no linear term). The doubling formula simplifies to $\lambda = 3x_1^2/(2y_1)$, saving one multiplication per doubling.

Parameter 3: The Coefficient b

$$b = 7$$

The curve equation is $y^2 = x^3 + 7$. The value $b = 7$ is the smallest integer producing a curve with the required group order.

Discriminant check: $4(0)^3 + 27(7)^2 = 1323 \neq 0$. ✓

Parameter 4: The Generator Point G

The generator (or **base point**) has coordinates:

$x_G =$ `79BE667E F9DCBBAC 55A06295 CE870B07`
`029BFCDB 2DCE28D9 59F2815B 16F81798`

$y_G =$ `483ADA77 26A3C465 5DA4FBFC 0E1108A8`
`FD17B448 A6855419 9C47D08F FB10D4B8`

This point was chosen deterministically as the point with the **smallest x-coordinate** that generates a group of the correct order. This "nothing up my sleeve" construction means nobody chose G to have a hidden backdoor.

Parameter 5: The Group Order n

$n =$ `FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE`
`BAAEDCE6 AF48A03B BFD25E8C D0364141`

This is the number of points in the group: $nG = \mathcal{O}$. Crucially, n is **prime**. By Corollary 4.5, this means the group is cyclic and every non-identity point is a generator—there are no small subgroups for an attacker to exploit.

Parameter 6: The Cofactor h

$$h = \frac{\#E(\mathbb{F}_p)}{n} = 1$$

The cofactor $h = 1$ means *every* point on the curve (except $\mathcal{O}$) has order n. Compare this to Curve25519, which has $h = 8$: its group contains a subgroup of order 8, requiring protocols to explicitly guard against small-subgroup attacks.

Parameter	Value	Role
p	$2^{256} - 2^{32} - 977$	Defines the field $\mathbb{F}_p$
a	0	Simplifies curve equation (Koblitz)
b	7	Completes curve: $y^2 = x^3 + 7$
G	(see hex above)	Generator point
n	(see hex above)	Group order (prime)
h	1	Cofactor (no subgroup attacks)

8.3 The "Nothing Up My Sleeve" Controversy

★ Satoshi's Insight: Why Not NIST P-256?

Satoshi chose secp256k1 over the more widely used NIST P-256 curve (secp256**r**1). This choice was prescient.

NIST P-256's coefficient $a = -3$ and its b value was derived from the SHA-1 hash of a seed:

```
c49d3608 86e70493 6a6678e1 139d26b7 819f7e90
```

Nobody knows where this seed came from. The concern is that whoever chose it might have searched for a seed that produces a curve with a hidden weakness—a "backdoor." This concern was amplified after the Dual_EC_DRBG scandal (2013), where the NSA was found to have inserted a backdoor into a different elliptic curve standard.

secp256k1, by contrast, has $a = 0$ and $b = 7$ chosen for efficiency, with G selected as the point with smallest x-coordinate. There is no unexplained random seed. The parameters are fully transparent.

▷ Steiner's Lens: Transparency as Epistemological Virtue

In GA2 (Ch. 7–8), Steiner argues that genuine knowledge requires *transparency*—the thinker must be able to follow every step from premises to conclusions, with nothing hidden or assumed on authority. A proof is valid not because an authority certifies it, but because any thinking person can verify each step independently. Knowledge that depends on an opaque source is not knowledge at all; it is belief.

The "nothing up my sleeve" principle in cryptographic parameter selection is Steiner's epistemological standard applied to engineering. NIST P-256 asks you to *trust* that an unexplained seed does not conceal a backdoor. secp256k1 asks you to *verify*: $a = 0$ for efficiency, $b = 7$ as the smallest valid value, G chosen deterministically as the point with smallest x-coordinate. Every parameter can be independently checked. The curve's security rests on mathematical proof, not institutional trust.

This is why Satoshi's choice of secp256k1 over the NIST curves was not merely a technical preference. It was an epistemological commitment: the security of the system should be *knowable*, not merely *believed*.

✓ Key Takeaways

- **secp256k1** = SEC standard, prime field, 256 bits, Koblitz, first curve.
- Six parameters: p (field prime), $a = 0$, $b = 7$ (curve equation), G (generator), n (group order), $h = 1$ (cofactor).
- The prime $p = 2^{256} - 2^{32} - 977$ enables fast modular arithmetic.
- $a = 0$ (Koblitz) simplifies point doubling.
- Group order n is prime: no subgroup attacks. Cofactor $h = 1$: every point generates the full group.
- Parameters are **fully transparent**—no unexplained random seeds, unlike NIST P-256.

9

Key Generation and the ECDLP

▸ The Story

We now have everything we need to understand how Bitcoin creates cryptographic keys. The entire process rests on a single idea: multiplying a secret number by a public point is easy, but reversing this multiplication is impossibly hard.
This chapter brings together modular arithmetic, group theory, and elliptic curves to explain the most important operation in Bitcoin: turning a private key into a public key.

9.1 Generating a Private Key

Definition 9.1 (Bitcoin Private Key). A **private key** is a randomly chosen integer d in the range $\{1, 2, \ldots, n-1\}$, where n is the group order of secp256k1.

That's it. A private key is just a random number—256 bits of entropy. In hexadecimal, it looks like:

```
e8f32e72 3de6dddf fcd0c9df 9d0ab4a6
f0e5a76a 0f5c67c7 d8f61d50 b37b2e29
```

The security requirement is that d must be chosen **uniformly at random**. If an attacker can predict or narrow down the possible values of d, they can steal the funds. This is why cryptographic random number generators (`/dev/urandom` on Linux, `crypto.getRandomValues()` in browsers) are essential—never use `rand()` or `Math.random()`.

9.2 Computing the Public Key: Scalar Multiplication

Definition 9.2 (Bitcoin Public Key). Given a private key d and the generator point G, the **public key** is:

$$Q = dG = \underbrace{G + G + \cdots + G}_{d \text{ times}}.$$

This is **scalar multiplication**: adding the point G to itself d times. The result Q is a point on the curve $E(\mathbb{F}_p)$.

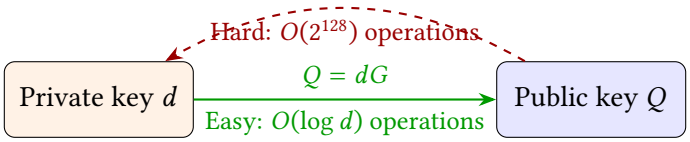


Figure 9.1: Key generation is a one-way function. Computing $Q = dG$ is efficient. Recovering d from Q and G is the ECDLP—believed computationally infeasible.

9.3 The Double-and-Add Algorithm

A private key d can be up to 2^{256}. We cannot literally add G to itself d times—that would take 2^{256} additions, which is more operations than atoms in the observable universe.

Instead, we use **double-and-add** (the additive analog of the square-and-multiply algorithm from Chapter 2):

1. Write d in binary: $d = (d_{255}\, d_{254} \cdots d_1\, d_0)_2$.
2. Start with $R = \mathcal{O}$ (the identity point).
3. For each bit from left to right:
   - **Double**: $R \leftarrow 2R$.
   - If the current bit is 1: **Add**: $R \leftarrow R + G$.
4. Return $R = dG$.

Example 9.1. Compute $42G$ using double-and-add. First, $42 = (101010)_2$.

Bit	Action	Running value of R
1	Double, Add G	$R = \mathcal{O} \to 2\mathcal{O} = \mathcal{O} \to \mathcal{O} + G = G$
0	Double	$R = 2G$
1	Double, Add G	$R = 4G \to 4G + G = 5G$
0	Double	$R = 10G$
1	Double, Add G	$R = 20G \to 20G + G = 21G$
0	Double	$R = 42G$

Total: 6 doublings + 3 additions = 9 operations, rather than 42 additions.

For a 256-bit key: at most 256 doublings + 256 additions = **512 elliptic curve operations**. This takes milliseconds on modern hardware.

Why This Matters: From Square-and-Multiply to Double-and-Add

In Chapter 2, we learned square-and-multiply for computing $a^n \bmod p$ efficiently. Double-and-add is exactly the same algorithm, but in additive notation:

	Multiplicative	**Additive (EC)**
Operation	a^n	nP
Main step	Square: $x \to x^2$	Double: $P \to 2P$
Conditional step	Multiply: $x \to x \cdot a$	Add: $P \to P + G$
Complexity	$O(\log n)$	$O(\log n)$

Every fast algorithm in elliptic curve cryptography—key generation, signing, verification—uses double-and-add at its core.

9.4 The ECDLP: Why You Can't Go Backward

Definition 9.3 (Elliptic Curve Discrete Logarithm Problem (ECDLP) — Restated)**.** Given an elliptic curve E over $\mathbb{F}_p$, a generator G of order n, and a point $Q \in E(\mathbb{F}_p)$, the **ECDLP** is: find the integer $d \in \{1, \ldots, n-1\}$ such that $Q = dG$.

In Chapter 5, we studied three attacks on the discrete logarithm problem:

Algorithm	**Time**	**For secp256k1 ($n \approx 2^{256}$)**
Brute force	$O(n)$	2^{256} operations
Baby-step giant-step	$O(\sqrt{n})$	2^{128} operations
Pollard's rho	$O(\sqrt{n})$	2^{128} operations

The best known attack on the ECDLP for a general elliptic curve over $\mathbb{F}_p$ is Pollard's rho, requiring $O(\sqrt{n}) \approx 2^{128}$ operations. Crucially, unlike the classical DLP in $\mathbb{F}_p^*$, there is **no index calculus attack** for elliptic curves. This is why ECC achieves the same security as RSA with much smaller keys.

Theorem 9.1 (Security of secp256k1)**.** *Recovering a secp256k1 private key from a public key requires approximately $\sqrt{n} \approx 2^{128}$ elliptic curve group operations. At 10^{12} (one trillion) operations per second, this would take approximately*

$$\frac{2^{128}}{10^{12} \times 365.25 \times 24 \times 3600} \approx 10^{19} \text{ years.}$$

The age of the universe is approximately 1.4×10^{10} years.

9.5 Public Key Encoding and Compression

A public key $Q = (x, y)$ consists of two 256-bit coordinates. There are two standard encodings:

Uncompressed format (65 bytes):

$$04 \parallel x \text{ (32 bytes)} \parallel y \text{ (32 bytes)}$$

The prefix 04 indicates an uncompressed key.

Compressed format (33 bytes):

$$02 \text{ or } 03 \parallel x \text{ (32 bytes)}$$

The prefix is 02 if y is even, 03 if y is odd.

Theorem 9.2 (Public Key Compression). *A compressed public key (33 bytes) contains sufficient information to recover the full public key (65 bytes).*

Proof. Given x, compute $c = x^3 + 7 \bmod p$. Since $p \equiv 3 \pmod 4$, the square root is $y = c^{(p+1)/4} \bmod p$. The two possible y-values are y and $p - y$, which have opposite parity. The prefix byte (02 or 03) selects the correct one. □

This compression saves 32 bytes per public key. In Bitcoin, every transaction output that specifies a public key benefits from this space savings. Taproot (BIP 340) goes further, using **x-only public keys** (32 bytes) by always choosing the y-coordinate with even parity.

★ Satoshi's Insight: Lost Coins

"Lost coins only make everyone else's coins worth slightly more. Think of it as a donation to everyone."
— Satoshi Nakamoto, BitcoinTalk, June 2010

Because the ECDLP is hard, if someone loses their private key d, there is no way to recover it from the public key $Q = dG$. The coins controlled by that key are gone forever. Satoshi understood this not as a bug but as a feature: lost coins reduce the supply, making remaining coins slightly more valuable.
An estimated 3–4 million BTC (out of 21 million total) are believed to be permanently lost, including approximately 1 million BTC in wallets attributed to Satoshi.

9.6 Address Derivation (Overview)

A Bitcoin address is derived from a public key through a sequence of hash functions:

$$\text{Address} = \text{Base58Check}(\text{RIPEMD-160}(\text{SHA-256}(Q))).$$

The double hashing provides several benefits:

- **Shorter output**: A 160-bit hash is shorter than a 256-bit public key coordinate.
- **Quantum resistance hedge**: Even if someone could solve the ECDLP (e.g., with a quantum computer), they would still need to find a preimage of RIPEMD-160 to go from an address back to a public key.
- **Checksum**: Base58Check encoding includes an error-detecting checksum.

Modern Bitcoin addresses (starting with bc1) use Bech32 encoding with a different hash scheme, but the principle is the same: one-way functions protect the public key until it is revealed in a spending transaction.

▷ Steiner's Lens: Self-Determined Identity

In GA4 (Ch. 9), Steiner develops the concept of the free individual: one whose actions flow from their own moral intuitions rather than from external compulsion. Such an individual is *self-determined*—their identity is constituted by their own inner activity, not assigned by an authority.

A Bitcoin keypair embodies this structure with mathematical precision. The private key d is chosen by the individual alone—no registration authority, no identity provider, no central database. The public key $Q = dG$ is derived deterministically from d. The relationship between d and Q is constituted entirely by the mathematics of the elliptic curve: no institution mediates, no permission is required, no record is kept.

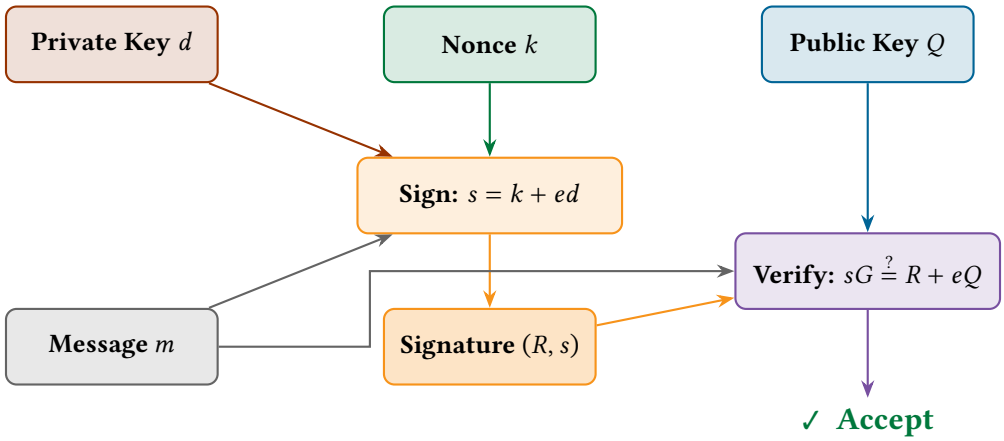
This is identity as Steiner conceived it: arising from the individual's own activity (the choice of d), expressed publicly through its consequences (Q), and secured not by institutional power but by the objective structure of mathematical reality (the ECDLP). The individual who holds a private key is sovereign in the strictest sense: no force in the universe—short of breaking the ECDLP—can impersonate them or revoke their identity.

✓ Key Takeaways

- A **private key** is a random integer $d \in \{1, \ldots, n-1\}$. A **public key** is $Q = dG$.
- **Double-and-add** computes dG in $O(\log d)$ operations—at most 512 EC operations for a 256-bit key.
- The **ECDLP** (recovering d from Q) requires $\sim 2^{128}$ operations—far beyond any computer.
- Public keys can be **compressed** from 65 to 33 bytes using the curve equation and a parity bit.
- Bitcoin addresses add an extra layer of protection through hashing.

Part IV

Signatures



Prove you know the secret, without revealing it.

"What is needed is an electronic payment system based on cryptographic proof instead of trust."

— Satoshi Nakamoto

10

ECDSA — The Original Bitcoin Signature

▸ The Story

A digital signature is a mathematical proof that the owner of a private key authorized a specific message—without revealing the key itself. Every Bitcoin transaction requires at least one signature. From Satoshi's genesis block to the latest Taproot spend, the act of moving bitcoin is the act of producing a valid signature.
The original Bitcoin protocol uses ECDSA: the Elliptic Curve Digital Signature Algorithm. It is elegant, subtle, and—as history has shown—dangerously unforgiving of implementation mistakes.

10.1 What a Digital Signature Must Do

A digital signature scheme has three algorithms:

1. **Key generation**: produce a private key d and public key $Q = dG$.
2. **Signing**: given a message m and private key d, produce a signature σ.
3. **Verification**: given m, σ, and the public key Q, output ACCEPT or REJECT.

The security requirements:

- **Correctness**: a signature produced with the correct key always verifies.
- **Unforgeability**: nobody can produce a valid signature without knowing d.
- **Non-repudiation**: the signer cannot deny having signed (since only they know d).

★ Satoshi's Insight: A Chain of Digital Signatures

"We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin."

— Satoshi Nakamoto, Bitcoin Whitepaper, Section 2

This is Bitcoin's core data structure. Every transfer of value is a signature. Understanding ECDSA means understanding how ownership moves through the Bitcoin network.

10.2 The ECDSA Signing Algorithm

Definition 10.1 (ECDSA Signing). **Input**: message m, private key d.

1. Compute the message hash: $z = H(m)$, where H is SHA-256 (truncated to the bit-length of n if necessary).
2. Choose a random **nonce** $k \in \{1, \dots, n-1\}$.
3. Compute the **ephemeral point**: $R = kG$.
4. Set $r = x_R \bmod n$ (the x-coordinate of R, reduced modulo n). If $r = 0$, go back to step 2.
5. Compute $s = k^{-1}(z + rd) \bmod n$. If $s = 0$, go back to step 2.
6. The signature is the pair (r, s).

Each signature is 64 bytes: r (32 bytes) and s (32 bytes).
Let us unpack what each step does:

- **Step 1**: Hashing the message ensures that the signature depends on the message content, regardless of message length.
- **Step 2**: The nonce k is a one-time random number. It must be truly random and **never reused**. We will see why shortly.
- **Step 3**: $R = kG$ is a "commitment" to the nonce, analogous to committing to a random point before revealing it.
- **Step 4**: Only the x-coordinate of R enters the signature—the y-coordinate is discarded.
- **Step 5**: This is the heart of ECDSA. The formula $s = k^{-1}(z + rd)$ binds together the message hash z, the nonce commitment r, and the private key d. The modular inverse k^{-1} (which we learned to compute in Chapter 2) ensures the equation can be "unwound" during verification.

10.3 The ECDSA Verification Algorithm

Definition 10.2 (ECDSA Verification). **Input**: message m, signature (r, s), public key Q.

1. Compute $z = H(m)$.
2. Compute $w = s^{-1} \bmod n$.
3. Compute $u_1 = zw \bmod n$ and $u_2 = rw \bmod n$.
4. Compute the point $P = u_1 G + u_2 Q$.
5. Accept the signature if $x_P \equiv r \pmod{n}$.

10.4 Why Verification Works

Theorem 10.1 (ECDSA Correctness). *If (r, s) is a valid ECDSA signature on m with private key d, then verification accepts.*

Proof. From signing: $s = k^{-1}(z + rd) \bmod n$.
Rearranging: $k = s^{-1}(z + rd) = s^{-1}z + s^{-1}rd = wz + wrd = u_1 + u_2 d \pmod{n}$.
Therefore:

$$\begin{aligned} P &= u_1 G + u_2 Q \\ &= u_1 G + u_2(dG) \\ &= (u_1 + u_2 d)G \\ &= kG \\ &= R. \end{aligned}$$

Since $P = R$, we have $x_P = x_R = r$, so verification accepts. □

Why This Matters: Every Verification Uses Modular Inverses

Notice that step 2 of verification computes $s^{-1} \bmod n$—the modular inverse from Chapter 2. Every time a Bitcoin node validates a transaction, it computes a modular inverse. Every time a full node syncs the entire blockchain, it computes millions of modular inverses. The Extended Euclidean Algorithm is not an abstract curiosity; it runs in the inner loop of Bitcoin's consensus engine.

10.5 The Nonce Catastrophe

The nonce k is the most dangerous variable in all of cryptography. If an attacker learns k for even a single signature, they can recover the private key.

Theorem 10.2 (Nonce Leakage ⇒ Key Recovery). *If an attacker knows the nonce k used in an ECDSA signature (r, s) on message m, they can compute the private key d.*

Proof. From the signing equation: $s = k^{-1}(z + rd) \bmod n$. Rearranging:

$$sk = z + rd \implies d = r^{-1}(sk - z) \bmod n.$$

All quantities on the right (r, s, k, z) are known to the attacker. □

Even worse: if the same nonce k is used for two different messages, the attacker doesn't even need to know k:

Theorem 10.3 (Nonce Reuse ⇒ Key Recovery). *If two signatures (r, s_1) and (r, s_2) use the same nonce k (for messages with hashes z_1, z_2), the private key can be recovered.*

Proof. The two signing equations are:

$$\begin{aligned} s_1 &= k^{-1}(z_1 + rd) \bmod n, \\ s_2 &= k^{-1}(z_2 + rd) \bmod n. \end{aligned}$$

Subtracting: $s_1 - s_2 = k^{-1}(z_1 - z_2) \bmod n$. Solving for k:

$$k = (z_1 - z_2)(s_1 - s_2)^{-1} \bmod n.$$

Then $d = r^{-1}(s_1 k - z_1) \bmod n$. □

Notice that both signatures share the same r value (since $r = x_{kG}$ and k is the same). Identical r values in two signatures are an immediate red flag.

Real-World Disasters

Incident	What Happened
PlayStation 3 (2010)	Sony used a *fixed* nonce (k was the same constant for every signature). Hackers recovered the private signing key, enabling piracy and homebrew software on every PS3 in existence.
Android Bitcoin Wallets (2013)	Android's `SecureRandom` had a bug that produced predictable output. Multiple Bitcoin wallets generated repeated nonces, and attackers drained funds by recovering private keys from the blockchain.
Blockchain.info (2014)	A server-side bug caused nonce reuse in the web wallet's signing code. Users lost funds before the issue was patched.

10.6 RFC 6979: Deterministic Nonces

The solution is to eliminate randomness from nonce generation entirely. RFC 6979 defines:

$$k = \text{HMAC-SHA256}(d, z)$$

where d is the private key and z is the message hash.

This is **deterministic**: the same message always produces the same nonce. But different messages produce different nonces (because z differs), and an attacker cannot predict k without knowing d.

Properties:

- **No nonce reuse**: different messages ⇒ different z ⇒ different k.
- **No RNG required**: signing doesn't need a random number generator at all.
- **Deterministic**: signing the same message twice produces the same signature, which aids testing.

10.7 Signature Malleability

There is one more subtlety. Given a valid signature (r, s), the pair $(r, n - s)$ is *also* a valid signature on the same message.

Why? Verification computes $P = u_1 G + u_2 Q$ using $w = s^{-1}$. Replacing s with $s' = n - s \equiv -s$ changes w to $(-s)^{-1} = -(s^{-1})$, which flips the signs of u_1 and u_2. But $(-u_1)G + (-u_2)Q = -(u_1 G + u_2 Q)$, and this negated point has the same x-coordinate as the original (since $-P = (x, -y)$). So verification still accepts.

This is called **signature malleability**: a third party can take a valid transaction and change its signature without invalidating it. This changes the transaction hash (txid), which caused problems in early Bitcoin (the "transaction malleability" bug).

BIP 62/BIP 146 fix: enforce the **low-s rule**: only accept signatures where $s \leq n/2$. Since exactly one of s and $n - s$ is $\leq n/2$, this pins down a unique canonical signature.

◇ From the Codebase: Constant-Time Signing

"MuSig2 is now available in libsecp256k1-zkp with a stable API. We've been extremely careful about the state machine—users can't accidentally reuse nonces or skip security checks."

— Jonas Nick, libsecp256k1 contributor

The nonce catastrophe shows why implementation quality matters as much as mathematical correctness. The libsecp256k1 library enforces RFC 6979 for ECDSA and implements careful state management for Schnorr and MuSig2 nonces. Every signing operation runs in constant time to prevent side-channel attacks.

▷ Steiner's Lens: The Signature as Moral Deed

In GA4 (Ch. 9–10), Steiner analyzes the structure of a genuinely free action. Such an act arises from *moral intuition*—the individual's direct perception of what a situation requires—and is expressed through a specific deed in the world. The deed is irrevocable: once performed, it enters reality and cannot be undone. It can be responded to, built upon, corrected—but the act itself stands.

A digital signature has exactly this structure. The signer perceives what needs to happen (a transfer of value), forms an intention (the transaction message), and performs an irrevocable act (the signature). The signature binds the signer's identity (d) to a specific commitment (m) through an irreversible mathematical operation. Once broadcast to the network, it cannot be recalled.

The nonce catastrophe reveals the moral weight of this structure. A careless signature—one made with a reused nonce—does not merely fail to authenticate. It *destroys* the signer's identity: the private key is exposed, the funds are lost, the mathematical person ceases to exist as a sovereign agent. In Steiner's framework, this is the consequence of acting without genuine attention—of performing the outward form of a deed without the inner content of awareness.

✓ Key Takeaways

- **ECDSA signing**: choose nonce k, compute $R = kG$, set $r = x_R$, compute $s = k^{-1}(z + rd) \bmod n$.
- **Verification**: compute $P = (zs^{-1})G + (rs^{-1})Q$ and check $x_P = r$.
- The **nonce catastrophe**: reusing k (or using a predictable k) leaks the private key. Real-world incidents have cost millions.
- **RFC 6979**: $k = \text{HMAC}(d, z)$ eliminates the need for randomness in signing.
- **Signature malleability**: (r, s) and $(r, n-s)$ are both valid; the low-s rule enforces a canonical form.

11

Schnorr Signatures — The Upgrade

▸ The Story

Claus-Peter Schnorr invented his signature scheme in 1989. It was simpler, more efficient, and more theoretically elegant than ECDSA. So why didn't Satoshi use it?
Because Schnorr patented it. U.S. Patent 4,995,082, filed in 1989, didn't expire until February 2008—just months before Bitcoin's whitepaper appeared. By the time Bitcoin launched in January 2009, the patent had expired, but the standardized curve libraries all used ECDSA (which had been developed specifically to avoid Schnorr's patent).
It took until November 2021—twelve years after Bitcoin's launch—for Schnorr signatures to arrive in Bitcoin through the **Taproot** upgrade (BIPs 340, 341, 342).

11.1 The Schnorr Signing Algorithm

Definition 11.1 (Schnorr Signing (BIP 340))**.** **Input**: message m, private key d.

1. Generate a deterministic nonce: $k = H_{\text{nonce}}(\text{aux} \,\|\, d \,\|\, m)$.
2. Compute the **nonce point**: $R = kG$. If y_R is odd, negate: $k \leftarrow n - k$ (so that R has even y).
3. Compute the **challenge**: $e = H_{\text{challenge}}(R \,\|\, Q \,\|\, m)$.
4. Compute $s = k + ed \bmod n$.
5. The signature is the pair (R, s), where R is encoded as its x-coordinate (32 bytes).

The notation H_{nonce} and $H_{\text{challenge}}$ refers to **tagged hashing**: $H_{\text{tag}}(x) = \text{SHA-256}(\text{SHA-256}(\text{tag}) \,\|\, \text{SHA-256}(\text{tag}) \,\|\, x)$. This domain separation ensures that a hash used for nonce generation cannot collide with a hash used for challenges.

11.2 The Schnorr Verification Algorithm

Definition 11.2 (Schnorr Verification (BIP 340))**.** **Input**: message m, signature (R, s), public key Q.

1. Compute $e = H_{\text{challenge}}(R \,\|\, Q \,\|\, m)$.
2. Accept if $sG = R + eQ$.

That's it. Verification is a single equation.

11.3 Why Verification Works

Theorem 11.1 (Schnorr Correctness). *If (R, s) is a valid Schnorr signature on m with private key d, then verification accepts.*

Proof. From signing: $s = k + ed \bmod n$, where $R = kG$ and $Q = dG$.

$$\begin{aligned} sG &= (k + ed)G \\ &= kG + edG \\ &= R + eQ. \qquad \checkmark \end{aligned}$$

□

Compare this to ECDSA's correctness proof, which required computing s^{-1} and tracking two scalar multiplications through substitutions. Schnorr's proof is three lines. This simplicity is not just aesthetic—it makes security proofs cleaner, implementations simpler, and bugs less likely.

11.4 ECDSA vs. Schnorr: A Comparison

Property	ECDSA	Schnorr (BIP 340)
Signature size	64 bytes	64 bytes
Signing formula	$s = k^{-1}(z + rd)$	$s = k + ed$
Verification	$u_1G + u_2Q$, check x	$sG \stackrel{?}{=} R + eQ$
Requires inverse?	Yes (k^{-1} and s^{-1})	No
Security proof	Complex	Simple (random oracle model)
Signature aggregation	Not possible	Yes (MuSig)
Batch verification	Inefficient	~2× speedup
Malleability	(r, s) and $(r, n - s)$	None (canonical by construction)

11.5 Batch Verification

When a Bitcoin node receives a new block, it must verify every signature in every transaction. With Schnorr, multiple signatures can be verified *simultaneously* using a technique called **batch verification**.

Given m signatures (R_i, s_i) on messages m_i with public keys Q_i, instead of checking each $s_iG = R_i + e_iQ_i$ individually, the verifier:

1. Chooses random weights $\alpha_1, \ldots, \alpha_m$.
2. Checks a single equation:

$$\left(\sum_{i=1}^{m} \alpha_i s_i\right) G = \sum_{i=1}^{m} \alpha_i R_i + \sum_{i=1}^{m} \alpha_i e_i Q_i.$$

The left side is one scalar multiplication. The right side is a multi-scalar multiplication, which can be computed much faster than m separate scalar multiplications using Pippenger's algorithm. The random weights α_i ensure that a batch containing even one invalid signature will fail with overwhelming probability.

The speedup: verifying m Schnorr signatures in a batch takes roughly the time of $\frac{m}{2} + 1$ individual verifications—approximately a **2× speedup** for large batches.

11.6 BIP 340: Bitcoin's Schnorr Specification

BIP 340 (co-authored by Pieter Wuille, Jonas Nick, and Tim Ruffing) specifies several design choices unique to Bitcoin:

- **x-only public keys**: Public keys are encoded as 32 bytes (just the x-coordinate) rather than 33 bytes (compressed). The y-coordinate is implicitly chosen to be even. This saves 1 byte per public key throughout the blockchain.
- **Tagged hashing**: All hash computations are domain-separated with tags (e.g., `"BIP0340/challenge"`, `"BIP0340/nonce"`) to prevent cross-protocol attacks.
- **No malleability**: Because R is encoded as its x-coordinate with implicit even y, and s is the unique value satisfying the signing equation, the signature is canonical. There is no second valid signature, unlike ECDSA's $(r, n - s)$ issue.
- **Taproot integration**: Schnorr signatures enable the Taproot spending rules (BIP 341), where a single signature can authorize a spend, hiding the fact that complex spending conditions (multisig, timelocks, hash locks) even exist.

Why This Matters: Taproot: Privacy Through Simplicity

Before Taproot, a multisig transaction was visibly different from a regular transaction on the blockchain—it used a different script type. With Schnorr and Taproot, a cooperative multisig spend looks identical to a single-signer spend: one public key, one signature.

This is a profound privacy improvement. When the common case (cooperative close of a Lightning channel, 2-of-3 corporate treasury spend) is indistinguishable from a simple payment, blockchain analysis becomes much harder. The mathematical simplicity of Schnorr—its linearity—is what makes this possible.

▷ Steiner's Lens: Simplicity as the Mark of Truth

In GA2 (Ch. 5), Steiner argues that when thinking grasps a concept truly, the result is experienced as *simple*—not because the subject matter is trivial, but because the mind has penetrated to its essence. Complexity is a sign that understanding is incomplete; genuine insight always simplifies.

Compare ECDSA's verification with Schnorr's. ECDSA requires computing s^{-1}, forming two scalar multiples u_1G and u_2Q, adding them, and checking an x-coordinate. Schnorr says: check whether $sG = R + eQ$. One equation. Three terms. The same security, achieved with radical economy.

Schnorr's correctness proof is three lines. Its security proof is clean enough to be taught in a first course on cryptography. Its linearity enables aggregation, batch verification, and Taproot—none of which are possible with ECDSA. This is not a coincidence. Schnorr's signature scheme penetrates to the *essence* of what a digital signature must do, and the simplicity of the result is the evidence.

Steiner would say: the mind that conceived $s = k + ed$ perceived the concept of "digital signature" more completely than the mind that conceived $s = k^{-1}(z + rd)$. The simpler formula is not a shortcut. It is a deeper seeing.

✓ Key Takeaways

- **Schnorr signing**: $s = k + ed$; **verification**: $sG = R + eQ$. No inverse needed.
- Schnorr is **simpler**, has a **cleaner security proof**, and is **not malleable**.
- **Batch verification** gives ~2× speedup for block validation.
- **BIP 340** uses x-only public keys (32 bytes) and tagged hashing.
- Schnorr's **linearity** ($s = k + ed$) enables signature aggregation (MuSig) and Taproot's privacy benefits.
- Schnorr signatures arrived in Bitcoin with **Taproot** (November 2021), twelve years after launch.

12

MuSig2 — Many Keys, One Signature

▸ The Story

Schnorr's linearity—the fact that $s = k + ed$ is a linear equation—opens a door that ECDSA keeps firmly shut. If two people each compute their own partial signature and add them together, the result is a valid signature for the *sum* of their public keys. This is signature aggregation, and it is the foundation of multisignature schemes, Lightning Network channels, and advanced Bitcoin scripts. But the naive approach has a devastating flaw. Fixing that flaw—securely and efficiently—is the subject of MuSig and its successor, MuSig2.

12.1 The Dream: Signature Aggregation

Consider three signers—Alice, Bob, and Carol—with key pairs (d_A, Q_A), (d_B, Q_B), (d_C, Q_C). They want to create a **3-of-3 multisignature**: all three must agree to authorize a transaction, but the result should look like a single signature from a single key.

If we naively define the aggregate public key as $\tilde{Q} = Q_A + Q_B + Q_C$, the hope is:

1. Each signer i picks nonce k_i, computes $R_i = k_i G$, and shares R_i.
2. The aggregate nonce is $\tilde{R} = R_A + R_B + R_C$.
3. Each signer computes a partial signature: $s_i = k_i + e d_i$.
4. The aggregate signature is $\tilde{s} = s_A + s_B + s_C$.
5. Verification: $\tilde{s}G = \tilde{R} + e\tilde{Q}$ holds because everything is linear.

This works—if all participants are honest.

12.2 The Rogue Key Attack

◇ From the Codebase: Why Naive Aggregation Fails

"You might think multi-signatures are simple: just add public keys together and add signatures together. But this creates a devastating attack where one party can control the aggregate key."

— Tim Ruffing, libsecp256k1 contributor

The rogue key attack shows that the simplest approach to multi-signatures is fatally flawed. The MuSig family of protocols solves this with a careful key aggregation step.

Here is the attack. Suppose Alice has public key Q_A, and the adversary Mallory claims to have public key Q_M. In naive aggregation, the aggregate key is $\tilde{Q} = Q_A + Q_M$.

But Mallory doesn't announce her *real* public key. Instead, she announces:

$$Q'_M = Q_M - Q_A.$$

Now the aggregate key is:

$$\tilde{Q} = Q_A + Q'_M = Q_A + (Q_M - Q_A) = Q_M.$$

Mallory controls the aggregate key entirely! She can sign for $\tilde{Q}$ without Alice's participation. Alice's key has been canceled out.

12.3 MuSig Key Aggregation

The MuSig protocol fixes the rogue key attack by multiplying each public key by a **key aggregation coefficient**:

Definition 12.1 (MuSig Key Aggregation). Given public keys $Q_1, \ldots, Q_m$, define the aggregate public key as:

$$\tilde{Q} = \sum_{i=1}^{m} a_i Q_i, \qquad \text{where } a_i = H_{\text{agg}}(L, Q_i)$$

and $L = (Q_1, Q_2, \ldots, Q_m)$ is the ordered list of all public keys.

The coefficients a_i depend on *all* the public keys, so Mallory cannot choose a key that cancels Alice's contribution: changing Q_M would change a_A and a_M as well, making the cancellation impossible.

12.4 From MuSig1 to MuSig2: The Round Problem

The original MuSig protocol (MuSig1, published 2018) requires **three rounds** of communication:

1. Each signer commits to their nonce: sends $\text{Com}_i = H(R_i)$.
2. Each signer reveals their nonce: sends R_i.
3. Each signer sends their partial signature s_i.

The commitment round prevents an attack (due to Wagner) where an adversary who sees all nonces before choosing their own can forge signatures. But three rounds are painful in practice:

⋄ From the Codebase: Two Rounds Is Qualitatively Different

"Three rounds is painful for real applications. Hardware wallets, cold storage, geographically distributed signers—every round adds latency and complexity. Two rounds is qualitatively different: you can pre-share the first round."

— Jonas Nick, Blockstream

The MuSig2 breakthrough eliminates the commitment round entirely, reducing signing to two rounds. This makes the protocol practical for hardware wallets, Lightning Network channels, and any setting where network round-trips are expensive.

12.5 The MuSig2 Protocol

MuSig2 (published 2020 by Jonas Nick, Tim Ruffing, and Yannick Seurin) achieves two-round signing through a clever trick: each signer generates **multiple nonces** instead of one.

Definition 12.2 (MuSig2 Signing Protocol). Let the signers have aggregate public key $\tilde{Q} = \sum a_i Q_i$.

Round 1 (can be pre-computed):

- Each signer i generates two nonce pairs: $(k_{i,1}, R_{i,1})$ and $(k_{i,2}, R_{i,2})$ where $R_{i,j} = k_{i,j}G$.
- Each signer broadcasts $(R_{i,1}, R_{i,2})$.

Round 2 (after the message m is known):

- Compute the **binding factor**: $b = H_{\text{bind}}(R_{1,1}, R_{1,2}, \ldots, R_{m,1}, R_{m,2}, m)$.
- Each signer's effective nonce: $R_i = R_{i,1} + b \cdot R_{i,2}$.
- Aggregate nonce: $\tilde{R} = \sum R_i$.
- Challenge: $e = H_{\text{challenge}}(\tilde{R} \,\|\, \tilde{Q} \,\|\, m)$.
- Each signer computes: $s_i = (k_{i,1} + b \cdot k_{i,2}) + e \cdot a_i \cdot d_i \bmod n$.
- Aggregate signature: $\tilde{s} = \sum s_i$.
- Final signature: $(\tilde{R}, \tilde{s})$.

The key insight is the **binding factor** b:

⋄ From the Codebase: The Binding Factor

"The binding factor b depends on the message. An attacker who sees nonces can't pre-compute how to combine them maliciously because they don't yet know which message will be signed."

— Jonas Nick, Blockstream

By making the linear combination of nonces depend on the message (through b), MuSig2 ensures that an adversary who sees pre-shared nonces (Round 1) cannot exploit them—they would need to know the message in advance to mount Wagner's attack.

12.6 Security Properties

MuSig2 achieves several important security properties:

- **Unforgeability**: No coalition of fewer than m signers can produce a valid signature (under the one-more discrete logarithm assumption).
- **Key indistinguishability**: The aggregate public key $\tilde{Q}$ is indistinguishable from a single-signer public key. On the blockchain, a MuSig2 multisig looks identical to a regular Schnorr signature.
- **Concurrent security**: Multiple signing sessions can be active simultaneously without compromising security—critical for Lightning Network nodes that handle many channels.
- **Nonce pre-sharing**: Round 1 can happen before the message is known, reducing interactive signing to a single round in practice.

Why This Matters: MuSig2 in Bitcoin Today

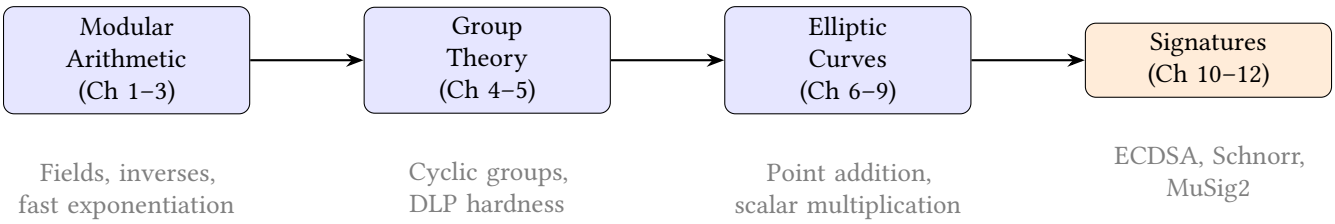
MuSig2 is already being used in production:

- **Lightning Network**: Channel funding transactions use 2-of-2 MuSig2, saving block space and improving privacy.
- **Corporate custody**: Multi-party corporate treasuries can use m-of-m MuSig2 for secure, private fund management.
- **Taproot keypath spends**: Any Taproot output can use MuSig2 internally while appearing as a single-key output on the blockchain.

For threshold signatures (t-of-m where $t < m$), the natural next step is **FROST** (Flexible Round-Optimized Schnorr Threshold signatures)—a topic for a future study guide.

12.7 Putting It All Together

We can now trace the full mathematical journey:



▷ Steiner's Lens: Community of Thinking

In GA4 (Ch. 9), Steiner argues that free individuals, each acting out of genuine moral intuition, do not collide—they harmonize. Two people who truly perceive what a situation requires will arrive at compatible actions, not because they obey the same external rule, but because they perceive the same moral reality from their individual standpoints. Freedom does not produce chaos. It produces a higher-order coordination than any imposed authority could achieve.

MuSig2 is the mathematical expression of this idea. Each signer acts independently—generating their own nonces, computing their own partial signature—yet the aggregate is a coherent whole: a single valid Schnorr signature indistinguishable from one produced by a solo signer. No central coordinator dictates the outcome. The protocol's algebraic structure ensures that independent contributions compose into a unified result.

This is not cooperation through coercion or compromise. It is cooperation through *shared mathematical structure*: each participant follows the same algebraic laws, and the linearity of Schnorr signatures guarantees that their independent actions combine harmoniously. Steiner would recognize this as an image of free community—individuals united not by external authority but by their common relationship to an objective reality.

✓ Key Takeaways

- **Naive aggregation** fails: the rogue key attack lets an adversary cancel other signers' keys.
- **MuSig** fixes this with key aggregation coefficients: $\tilde{Q} = \sum a_i Q_i$.
- **MuSig1** requires 3 rounds (commit, reveal, sign); **MuSig2** requires only 2 rounds.
- The **binding factor** b (message-dependent) prevents Wagner's attack without a commitment round.
- The aggregate signature $(\tilde{R}, \tilde{s})$ is indistinguishable from a single-signer Schnorr signature on the blockchain.
- MuSig2 is already in production use for Lightning, corporate custody, and Taproot spends.

Epilogue: What Satoshi Built

★ Satoshi's Insight: The Root Problem

"The root problem with conventional currency is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust."

— Satoshi Nakamoto, P2P Foundation, February 11, 2009

This is the sentence that started everything. The entire mathematical edifice we have built in this guide—from clock arithmetic to MuSig2—exists to replace trust with proof.

Let us trace the path one final time.

When you send bitcoin, your wallet performs a sequence of operations that traverses every chapter of this guide:

1. Your **private key** d is a random element of $\mathbb{Z}_n$ (**modular arithmetic**, Ch 1–3).
2. Your **public key** $Q = dG$ is computed by **scalar multiplication** on the elliptic curve $E : y^2 = x^3 + 7$ over the finite field $\mathbb{F}_p$ (**elliptic curves**, Ch 6–9). The double-and-add algorithm, the additive version of square-and-multiply from Ch 2, performs this in milliseconds.
3. The security of your key rests on the **ECDLP**: given Q and G, nobody can find d. This hardness comes from the **group structure** of the curve (**groups and DLP**, Ch 4–5), specifically the absence of index calculus attacks on elliptic curves.
4. To authorize a transaction, your wallet produces a **digital signature**—either ECDSA (Ch 10) or, since Taproot, a Schnorr signature (Ch 11). The signature binds your private key to the transaction message through a carefully constructed algebraic equation.
5. If the transaction involves multiple parties (a Lightning channel, a corporate treasury), **MuSig2** (Ch 12) produces a single aggregate signature that is indistinguishable from a solo spend—providing both efficiency and privacy.
6. Every node on the network **verifies** your signature using only your public key and the transaction data. No trust is required. The verification equation—$sG = R + eQ$ for Schnorr, or the more involved ECDSA check—is a purely mathematical statement that either holds or doesn't.

The mathematics is deep, but the message is simple: *cryptographic proof replaces institutional trust.*

A random number, multiplied by a special point on a special curve over a special prime field, creates a public identity. An algebraic equation, satisfied by a signature that only the key-holder can produce, authorizes the transfer of value. And the hardness of one particular problem—the elliptic curve discrete logarithm problem—ensures that the system is secure against any adversary operating within the laws of physics.

Jonas Nick and his colleagues at Blockstream continue to push this mathematics forward. MuSig2 brought efficient multi-signatures to Bitcoin. FROST will bring flexible threshold signatures. Adaptor signatures enable atomic cross-chain swaps. Formal verification is being applied to the libsecp256k1 codebase itself.

The curve that powers Bitcoin—secp256k1, with its six carefully chosen parameters—is a fixed mathematical object. But the protocols built on top of it continue to evolve, each new construction unlocking capabilities that were impossible just years earlier.

▷ Steiner's Lens: The Completed Circle

In GA4 (Ch. 12), Steiner describes the arc of genuine knowing: thinking begins with an immediate perception of a concept's reality, works through the concept's connections and consequences, and finally returns—enriched—to a deepened grasp of the original perception. The circle does not close where it started. It closes *above* where it started, having gained the content of everything traversed. This study guide has traced such a circle. We began with the simplest idea in mathematics: the remainder when one number divides another. From that seed grew fields, groups, curves, and finally signatures—each layer building on the last with logical necessity. The final result—a 64-byte Schnorr signature that authorizes the transfer of value across a trustless network—is implicit in the first chapter's division algorithm, just as the oak is implicit in the acorn.

But the circle does not close as mere abstraction. It closes as a *deed*: a signature that moves real value in the real world. Steiner's epistemology insists that thinking is not opposed to reality—it *is* reality, grasped from within. The mathematics of elliptic curves is not a model of Bitcoin's security. It is that security. The thinking and the reality are one.

From clock arithmetic to Schnorr signatures,
from a 256-bit random number to a global financial network—
this is the mathematics that makes Bitcoin work.

Exercises

Exercises are graded by difficulty: ⋆ = routine, ⋆⋆ = moderate, ⋆ ⋆ ⋆ = challenging.

Chapter 1: Clock Arithmetic

Exercise 1.1 (⋆). Compute the following without a calculator:

(a) $47 \bmod 7$

(b) $-3 \bmod 11$

(c) $100 \bmod 13$

(d) $2^{10} \bmod 7$

Exercise 1.2 (⋆). Write out the complete addition and multiplication tables for $\mathbb{Z}_5$.

Exercise 1.3 (⋆⋆). Prove that if $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $ac \equiv bd \pmod{n}$.

Exercise 1.4 (⋆⋆). Find all solutions to $3x \equiv 6 \pmod{9}$. Why is the answer not unique?

Exercise 1.5 (⋆⋆). For which values of n does $\mathbb{Z}_n$ have zero divisors (nonzero a, b with $ab \equiv 0$)? Prove your answer.

Exercise 1.6 (⋆ ⋆ ⋆). Prove that a has a multiplicative inverse modulo n if and only if $\gcd(a, n) = 1$.

Chapter 2: The Inverse Problem

Exercise 2.1 (⋆). Use the Extended Euclidean Algorithm to find $7^{-1} \bmod 31$. Verify your answer.

Exercise 2.2 (⋆). Compute $3^{-1} \bmod 17$ using Fermat's Little Theorem ($a^{-1} \equiv a^{p-2}$).

Exercise 2.3 (⋆⋆). Use the square-and-multiply algorithm to compute $5^{117} \bmod 19$. Show all steps.

Exercise 2.4 (⋆⋆). Prove that in $\mathbb{Z}_p$ (with p prime), $(-1)^2 = 1$ and $(-1) \neq 1$ (when $p > 2$). Conclude that $p - 1 \equiv -1 \pmod{p}$ is a square root of 1 distinct from 1.

Exercise 2.5 (⋆ ⋆ ⋆). Prove **Wilson's Theorem**: $(p-1)! \equiv -1 \pmod{p}$ for any prime p.

Hint: pair each element of $\{1, 2, \ldots, p-1\}$ with its multiplicative inverse. Which elements are their own inverse?

Exercise 2.6 (⋆ ⋆ ⋆). The Extended Euclidean Algorithm computes $\gcd(a, b)$ and integers x, y such that $ax + by = \gcd(a, b)$. Prove that the algorithm terminates in at most $\lfloor \log_\phi(\max(a, b)) \rfloor + 1$ steps, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio.

Hint: show that the remainders decrease at least as fast as the Fibonacci sequence decreases backward.

Chapter 3: The Finite World

Exercise 3.1 (⋆). Verify all 11 field axioms for $\mathbb{Z}_7$. Explicitly exhibit the additive and multiplicative inverses of each element.

Exercise 3.2 (⋆⋆). Show that $\mathbb{Z}_6$ is not a field by finding a specific axiom that fails.

Exercise 3.3 (⋆⋆). In $\mathbb{F}_{31}$, compute:

(a) $17 + 28$

(b) 17×28

(c) 28^{-1}

(d) $17/28$ (i.e., 17×28^{-1})

Exercise 3.4 (⋆⋆). Explain why the pseudo-Mersenne form $p = 2^{256} - c$ (with small c) allows fast modular reduction. Specifically, show that if $x = x_H \cdot 2^{256} + x_L$ (where $x_H < 2^{256}$ and $x_L < 2^{256}$), then $x \equiv x_L + c \cdot x_H \pmod{p}$.

Exercise 3.5 (⋆ ⋆ ⋆). Prove that a finite field of order n exists if and only if $n = p^k$ for some prime p and positive integer k. (You may assume the existence of finite fields of prime order.)

Hint: for "only if," consider the characteristic of the field.

Chapter 4: Groups

Exercise 4.1 (⋆). Verify that $(\mathbb{Z}_7^*, \times)$ is a group. What is its order? Find the order of each element.

Exercise 4.2 (⋆). Find all generators of $\mathbb{Z}_{11}^*$.

Exercise 4.3 (⋆⋆). Prove that in any group, the identity element is unique.

Exercise 4.4 (⋆⋆). Let G be a group of order 15. What are the possible orders of elements? Is G necessarily cyclic?

Hint: use Lagrange's theorem, then consider that $15 = 3 \times 5$.

Exercise 4.5 (⋆⋆). Prove that every group of prime order p is isomorphic to $\mathbb{Z}_p$.

Exercise 4.6 (⋆ ⋆ ⋆). Let G be a finite group and H, K subgroups with $\gcd(|H|, |K|) = 1$. Prove that $H \cap K = \{e\}$.

Chapter 5: The Discrete Logarithm Problem

Exercise 5.1 (⋆). In $\mathbb{Z}_{23}^*$, compute $5^x \bmod 23$ for $x = 1, 2, \ldots, 22$. Verify that 5 is a generator.

Exercise 5.2 (⋆⋆). Using baby-step giant-step, find x such that $3^x \equiv 13 \pmod{17}$. Show all baby steps and giant steps.

Exercise 5.3 (⋆⋆). Explain why baby-step giant-step requires $O(\sqrt{n})$ storage. Why is this a problem for $n \approx 2^{256}$?

Exercise 5.4 (⋆⋆). In Pollard's rho algorithm, explain why the expected number of steps before a collision is $O(\sqrt{n})$. Relate this to the birthday paradox.

Exercise 5.5 (⋆ ⋆ ⋆). Explain why index calculus works for $\mathbb{F}_p^*$ but not for elliptic curve groups. What property of $\mathbb{F}_p^*$ does it exploit, and why is this property absent for elliptic curves?

Chapter 6: Elliptic Curves Over the Reals

Exercise 6.1 (⋆). Verify that the point $(2, \sqrt{15})$ lies on the curve $y^2 = x^3 + 7$.

Exercise 6.2 (⋆). Check the discriminant condition for the curves:

(a) $y^2 = x^3 + 3x + 5$

(b) $y^2 = x^3 - 3x + 2$

(c) $y^2 = x^3 + 7$

Which are valid elliptic curves?

Exercise 6.3 (⋆⋆). On the curve $y^2 = x^3 - 7x + 10$, let $P = (1, 2)$ and $Q = (3, 4)$. Verify that P and Q are on the curve, then compute $P + Q$ using the addition formulas.

Exercise 6.4 (⋆⋆). On $y^2 = x^3 + 7$, compute $2P$ where $P = (1, \sqrt{8})$. (Use the doubling formula with $a = 0$.)

Exercise 6.5 (⋆⋆). Explain geometrically why the point addition operation on an elliptic curve is commutative.

Exercise 6.6 (⋆⋆⋆). Prove the addition formula $x_3 = \lambda^2 - x_1 - x_2$ by substituting the line $y = \lambda(x - x_1) + y_1$ into $y^2 = x^3 + ax + b$ and applying Vieta's formulas.

Chapter 7: Elliptic Curves Over Finite Fields

Exercise 7.1 (⋆). Using Euler's criterion, determine which of $\{1, 2, 3, 4, 5, 6\}$ are quadratic residues modulo 7.

Exercise 7.2 (⋆⋆). Find all points on $E : y^2 = x^3 + 2x + 3$ over $\mathbb{F}_7$. Count them and verify Hasse's bound.

Exercise 7.3 (⋆⋆). On $E : y^2 = x^3 + 7$ over $\mathbb{F}_{23}$, verify that $P = (1, 10)$ is on the curve, then compute $3P$ (i.e., $2P + P$).

Exercise 7.4 (⋆⋆). Explain why $p \equiv 3 \pmod 4$ makes square root computation easy. For $p = 23$, compute $\sqrt{8} \bmod 23$ using the formula $y = c^{(p+1)/4}$.

Exercise 7.5 (⋆⋆⋆). Prove Hasse's bound gives $\#E(\mathbb{F}_p) \in [p+1-2\sqrt{p},\ p+1+2\sqrt{p}]$. For $p = 2^{256} - 2^{32} - 977$, compute the width of this interval as a fraction of p.

Chapter 8: secp256k1

Exercise 8.1 (⋆). What does each component of the name "secp256k1" stand for?

Exercise 8.2 (⋆). List the six parameters of secp256k1 and state the role of each.

Exercise 8.3 (⋆⋆). Verify that $p = 2^{256} - 2^{32} - 977$ satisfies $p \equiv 3 \pmod 4$.
Hint: compute $2^{256} \bmod 4$, $2^{32} \bmod 4$, and $977 \bmod 4$.

Exercise 8.4 (⋆⋆). Explain why a cofactor of $h = 1$ eliminates small-subgroup attacks. What additional steps would be required if $h = 8$ (as in Curve25519)?

Exercise 8.5 (⋆ ⋆ ⋆). Research the Dual_EC_DRBG backdoor. Explain how a party who chose the generator point Q for the DRBG could predict its output. Why does secp256k1's deterministic generator choice avoid this risk?

Chapter 9: Key Generation

Exercise 9.1 (⋆). How many elliptic curve operations does double-and-add require to compute dG for a 256-bit key d? How does this compare to naive repeated addition?

Exercise 9.2 (⋆⋆). Use the double-and-add algorithm to compute $100P$. Write 100 in binary and trace through each step (don't compute actual coordinates—just express the result at each step in terms of P).

Exercise 9.3 (⋆⋆). A compressed public key is 33 bytes: a prefix byte (`02` or `03`) followed by the 32-byte x-coordinate. Describe the decompression algorithm step by step, explaining why $p \equiv 3 \pmod 4$ is useful.

Exercise 9.4 (⋆⋆). If a quantum computer could solve the ECDLP in polynomial time, would funds sent to a Bitcoin *address* (as opposed to a raw public key) be immediately vulnerable? Explain, referencing the hash layers in address derivation.

Exercise 9.5 (⋆ ⋆ ⋆). Pollard's rho applied to secp256k1 requires $O(2^{128})$ group operations. Suppose you have a cluster performing 10^{15} group operations per second. How many years would it take to break one key? Express your answer as a power of 10.

Chapter 10: ECDSA

Exercise 10.1 (⋆). In ECDSA, what are the two components of a signature? How many bytes is a standard signature?

Exercise 10.2 (⋆⋆). Suppose two ECDSA signatures (r, s_1) and (r, s_2) are observed on messages with hashes $z_1 = 42$ and $z_2 = 99$, with $s_1 = 17$ and $s_2 = 53$, all modulo $n = 101$. If the nonce was reused, recover k and then d (assuming $r = 37$).

Exercise 10.3 (⋆⋆). Explain why observing two signatures with the same r value is a red flag for nonce reuse.

Exercise 10.4 (⋆⋆). Prove that $(r, n - s)$ is also a valid ECDSA signature if (r, s) is. Why does BIP 62 require $s \leq n/2$?

Exercise 10.5 (⋆ ⋆ ⋆). In RFC 6979, $k = \text{HMAC-SHA256}(d, z)$. Explain why this is secure even though it is deterministic. Under what conditions would two signatures produce the same k? Is this a problem?

Chapter 11: Schnorr Signatures

Exercise 11.1 (⋆). Write the Schnorr verification equation. How many scalar multiplications does it require?

Exercise 11.2 (⋆). What is an "x-only public key" in BIP 340, and how many bytes does it save compared to a compressed public key?

Exercise 11.3 (⋆⋆). Prove Schnorr correctness: show that if $s = k + ed$ where $R = kG$ and $Q = dG$, then $sG = R + eQ$.

Exercise 11.4 (⋆⋆). Explain how batch verification of m Schnorr signatures achieves a ~2× speedup. What role do the random weights α_i play?

Exercise 11.5 (⋆⋆). Why is Schnorr's signing formula $s = k + ed$ called "linear," while ECDSA's $s = k^{-1}(z + rd)$ is not? What does linearity enable?

Exercise 11.6 (⋆ ⋆ ⋆). Explain why Schnorr signatures are not malleable (i.e., (R, s) is the unique valid signature for a given k), while ECDSA signatures are. Reference the role of R's encoding.

Chapter 12: MuSig2

Exercise 12.1 (⋆). Describe the rogue key attack in one paragraph. What happens if we aggregate public keys by simple addition?

Exercise 12.2 (⋆⋆). In MuSig key aggregation, $\tilde{Q} = \sum a_i Q_i$ where $a_i = H(L, Q_i)$. Explain why an adversary cannot perform a rogue key attack with this construction.

Exercise 12.3 (⋆⋆). Why does MuSig1 require three rounds while MuSig2 requires only two? What attack does the commitment round in MuSig1 prevent?

Exercise 12.4 (⋆⋆). In MuSig2, the binding factor is $b = H(\text{all nonces}, m)$. Why must b depend on the message m? What would go wrong if b were computed before the message was known?

Exercise 12.5 (⋆ ⋆ ⋆). Verify algebraically that a MuSig2 aggregate signature $(\tilde{R}, \tilde{s})$ passes Schnorr verification against the aggregate key $\tilde{Q}$. That is, show $\tilde{s}G = \tilde{R} + e\tilde{Q}$.

Exercise 12.6 (⋆ ⋆ ⋆). Compare MuSig2 (which achieves m-of-m) to FROST (which achieves t-of-m). Why is the threshold case harder? What additional mathematical machinery (e.g., polynomial interpolation) is needed?

Solutions

Chapter 1 Solutions

1.1. (a) $47 = 6 \times 7 + 5$, so $47 \bmod 7 = 5$. (b) $-3 + 11 = 8$, so $-3 \bmod 11 = 8$. (c) $100 = 7 \times 13 + 9$, so $100 \bmod 13 = 9$. (d) $2^{10} = 1024$. $1024 = 146 \times 7 + 2$, so $2^{10} \bmod 7 = 2$. (Alternatively: $2^3 = 8 \equiv 1$, so $2^{10} = (2^3)^3 \cdot 2 \equiv 1^3 \cdot 2 = 2$.)

1.2. Addition table for $\mathbb{Z}_5$:

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\times$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

1.3. Since $a \equiv b \pmod{n}$, we have $a = b + kn$ for some integer k. Similarly $c = d + \ell n$. Then $ac = (b + kn)(d + \ell n) = bd + (b\ell + dk + k\ell n)n \equiv bd \pmod{n}$.

1.4. $3x \equiv 6 \pmod{9}$ simplifies to $x \equiv 2 \pmod{3}$ (divide both sides and the modulus by $\gcd(3, 9) = 3$). The solutions in $\mathbb{Z}_9$ are $x \in \{2, 5, 8\}$. The answer is not unique because $\gcd(3, 9) = 3 > 1$, so 3 is not invertible modulo 9.

1.5. $\mathbb{Z}_n$ has zero divisors if and only if n is composite. If $n = ab$ with $1 < a, b < n$, then $a \cdot b = n \equiv 0 \pmod{n}$ but $a, b \neq 0$. If n is prime, then $\mathbb{Z}_n$ is a field (every nonzero element has an inverse), so there are no zero divisors.

1.6. ($\Rightarrow$) If a has an inverse b modulo n, then $ab \equiv 1 \pmod{n}$, so $ab = 1 + kn$ for some k, giving $ab - kn = 1$. This is a linear combination of a and n equaling 1, so $\gcd(a, n) = 1$.

($\Leftarrow$) If $\gcd(a, n) = 1$, the Extended Euclidean Algorithm finds x, y with $ax + ny = 1$. Reducing modulo n: $ax \equiv 1 \pmod{n}$, so x is the inverse of a.

Chapter 2 Solutions

2.1. Extended Euclidean Algorithm on $(31, 7)$: $31 = 4 \times 7 + 3$; $7 = 2 \times 3 + 1$; $3 = 3 \times 1 + 0$.

Back-substituting: $1 = 7 - 2 \times 3 = 7 - 2(31 - 4 \times 7) = 9 \times 7 - 2 \times 31$.

So $7^{-1} \equiv 9 \pmod{31}$. Check: $7 \times 9 = 63 = 2 \times 31 + 1 \equiv 1$. ✓

2.2. By Fermat's Little Theorem, $3^{-1} \equiv 3^{17-2} = 3^{15} \pmod{17}$. Compute: $3^2 = 9$, $3^4 = 81 \equiv 13$, $3^8 \equiv 13^2 = 169 \equiv 169 - 9 \times 17 = 16$, $3^{15} = 3^8 \times 3^4 \times 3^2 \times 3^1 = 16 \times 13 \times 9 \times 3$. $16 \times 13 = 208 \equiv 208 - 12 \times 17 = 4$. $4 \times 9 = 36 \equiv 2$. $2 \times 3 = 6$. So $3^{-1} \equiv 6 \pmod{17}$. Check: $3 \times 6 = 18 \equiv 1$. ✓

2.3. $117 = (1110101)_2$. Square-and-multiply for 5^{117} mod 19:

$5^1 = 5$. $5^2 = 25 \equiv 6$. $5^4 = 36 \equiv 17$. $5^8 = 17^2 = 289 \equiv 289 - 15 \times 19 = 4$. $5^{16} = 16$. $5^{32} = 256 \equiv 256 - 13 \times 19 = 9$. $5^{64} = 81 \equiv 81 - 4 \times 19 = 5$.

$117 = 64 + 32 + 16 + 4 + 1$, so $5^{117} = 5 \times 9 \times 16 \times 17 \times 5$. $5 \times 9 = 45 \equiv 7$. $7 \times 16 = 112 \equiv 112 - 5 \times 19 = 17$. $17 \times 17 = 289 \equiv 4$. $4 \times 5 = 20 \equiv 1$. So $5^{117} \equiv 1 \pmod{19}$. (This is consistent with Fermat: $5^{18} \equiv 1$, and $117 = 6 \times 18 + 9$; checking 5^9: $5^9 = 5^8 \times 5 = 4 \times 5 = 20 \equiv 1$. Actually $117/18 = 6.5$, so $117 = 6 \times 18 + 9$. We need 5^9 mod 19: $5^9 = 5^8 \times 5 \equiv 4 \times 5 = 20 \equiv 1$. ✓)

2.4. In $\mathbb{Z}_p$, $(-1)^2 = 1$ is immediate. For $p > 2$, $-1 \equiv p - 1 \neq 1$ since $p > 2$. So $\mathbb{Z}_p$ has at least two square roots of 1: namely 1 and $p - 1$. (In fact, these are the only square roots, since $x^2 = 1$ means $p \mid (x-1)(x+1)$, and p prime implies $p \mid (x-1)$ or $p \mid (x+1)$.)

2.5. Pair each $a \in \{1, \ldots, p-1\}$ with its inverse a^{-1}. The elements satisfying $a = a^{-1}$ (i.e., $a^2 \equiv 1$) are $a = 1$ and $a = p - 1$ (by Exercise 2.4). All other elements pair up: $a \neq a^{-1}$, so $(a)(a^{-1}) = 1$ in each pair. Therefore $(p-1)! = 1 \times (p-1) \times \prod_{\text{pairs}} 1 = p - 1 \equiv -1 \pmod{p}$.

2.6. Let r_i be the sequence of remainders. We show $r_i \geq F_{n-i}$ (Fibonacci numbers) where n is the total number of steps. Since $r_i > r_{i+1}$ and $r_i = q_{i+1} r_{i+1} + r_{i+2}$ with $q_{i+1} \geq 1$, we get $r_i \geq r_{i+1} + r_{i+2}$. The Fibonacci recurrence gives $r_i \geq F_{n-i}$, so $r_0 \geq F_n$. Since $F_n \sim \phi^n / \sqrt{5}$, we get $n \leq \log_\phi(r_0 \sqrt{5}) = O(\log_\phi(\max(a, b)))$.

Chapter 3 Solutions

3.1. In $\mathbb{Z}_7$: additive inverses: $-0 = 0, -1 = 6, -2 = 5, -3 = 4, -4 = 3, -5 = 2, -6 = 1$. Multiplicative inverses of nonzero elements: $1^{-1} = 1, 2^{-1} = 4$ ($2 \times 4 = 8 \equiv 1$), $3^{-1} = 5$ ($3 \times 5 = 15 \equiv 1$), $4^{-1} = 2, 5^{-1} = 3, 6^{-1} = 6$ ($6 \times 6 = 36 \equiv 1$). Closure, associativity, commutativity, distributivity follow from $\mathbb{Z}_7$ being a quotient of $\mathbb{Z}$. Identity elements: 0 for $+$, 1 for $\times$. All 11 axioms hold.

3.2. In $\mathbb{Z}_6$, the element 2 has no multiplicative inverse: $2 \times 0 = 0$, $2 \times 1 = 2$, $2 \times 2 = 4$, $2 \times 3 = 0$, $2 \times 4 = 2$, $2 \times 5 = 4$. None equal 1. Alternatively, $2 \times 3 = 0$ shows $\mathbb{Z}_6$ has zero divisors, violating the field axiom that every nonzero element has a multiplicative inverse.

3.3. In $\mathbb{F}_{31}$: (a) $17 + 28 = 45 \equiv 45 - 31 = 14$. (b) $17 \times 28 = 476 \equiv 476 - 15 \times 31 = 476 - 465 = 11$. (c) $28^{-1} \equiv 28^{29} \pmod{31}$. Shortcut: $28 \equiv -3$, so $28^{-1} \equiv (-3)^{-1} \equiv -(3^{-1})$. Now $3 \times 21 = 63 = 2 \times 31 + 1$, so $3^{-1} = 21$. Thus $28^{-1} = -21 \equiv 10$. (d) $17 \times 10 = 170 \equiv 170 - 5 \times 31 = 15$.

3.4. If $x < 2^{256}$, then $x \bmod p = x$. If $x = x_H \cdot 2^{256} + x_L$, then since $2^{256} \equiv c \pmod{p}$ (where $c = 2^{32} + 977$), we get $x \equiv c \cdot x_H + x_L \pmod{p}$. Since c is small ($\approx 2^{32}$), the product $c \cdot x_H$ has many fewer bits than a generic 256×256-bit multiplication, making the reduction much faster.

3.5. Let F be a finite field with $|F| = n$. The additive group $(F, +)$ is a finite abelian group; let p be the characteristic (smallest positive integer with $p \cdot 1 = 0$). Then p must be prime (if $p = ab$, then $(a \cdot 1)(b \cdot 1) = 0$ in a field forces $a \cdot 1 = 0$ or $b \cdot 1 = 0$, contradicting minimality). The prime subfield $\mathbb{F}_p \subseteq F$ makes F a vector space over $\mathbb{F}_p$ of some dimension k, so $|F| = p^k$.

Chapter 4 Solutions

4.1. $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$ under multiplication mod 7. Order: $|\mathbb{Z}_7^*| = 6$. Element orders: ord(1) = 1. ord(2): $2, 4, 1 \Rightarrow \text{ord}(2) = 3$. ord(3): $3, 2, 6, 4, 5, 1 \Rightarrow \text{ord}(3) = 6$. ord(4) = ord($2^2$): $4, 2, 1 \Rightarrow 3$. ord(5): $5, 4, 6, 2, 3, 1 \Rightarrow 6$. ord(6): $6, 1 \Rightarrow 2$.

4.2. Generators of $\mathbb{Z}_{11}^*$ are elements of order $\phi(11) = 10$. By Lagrange, possible orders divide 10: $\{1, 2, 5, 10\}$. Testing: ord(2): $2, 4, 8, 5, 10, 9, 7, 3, 6, 1 \Rightarrow \text{ord}(2) = 10$. Since there are $\phi(10) = 4$ generators, they are the elements g^k where $\gcd(k, 10) = 1$: $2^1 = 2$, $2^3 = 8$, $2^7 = 7$, $2^9 = 6$. Generators: $\{2, 6, 7, 8\}$.

4.3. Suppose e and e' are both identities. Then $e = e * e' = e'$ (using e' as identity for the first equality, e as identity for the second).

4.4. By Lagrange, element orders divide 15. Divisors of 15: $\{1, 3, 5, 15\}$. Yes, G is cyclic: since $15 = 3 \times 5$ and $\gcd(3, 5) = 1$, by the classification of finite abelian groups, $G \cong \mathbb{Z}_{15}$ (the only group of order 15 up to isomorphism).

4.5. Let $|G| = p$ (prime) and let $g \neq e$. By Lagrange, ord(g) divides p. Since ord(g) > 1 and p is prime, ord(g) $= p$. So $G = \langle g \rangle \cong \mathbb{Z}_p$.

4.6. Let $x \in H \cap K$. Then ord(x) divides $|H|$ and $|K|$ (by Lagrange in each subgroup). Since $\gcd(|H|, |K|) = 1$, ord(x) divides 1, so ord(x) $= 1$, meaning $x = e$.

Chapter 5 Solutions

5.1. $5^1 = 5, 5^2 = 2, 5^3 = 10, 5^4 = 4, 5^5 = 20, 5^6 = 8, 5^7 = 17, 5^8 = 16, 5^9 = 11, 5^{10} = 9, 5^{11} = 22, 5^{12} = 18, 5^{13} = 21, 5^{14} = 13, 5^{15} = 19, 5^{16} = 3, 5^{17} = 15, 5^{18} = 6, 5^{19} = 7, 5^{20} = 12, 5^{21} = 14, 5^{22} = 1$. All 22 nonzero elements appear, confirming 5 is a generator. ✓

5.2. Find x with $3^x \equiv 13 \pmod{17}$. Group order $n = 16$, so $m = \lceil \sqrt{16} \rceil = 4$.

Baby steps: $3^0 = 1, 3^1 = 3, 3^2 = 9, 3^3 = 10$.

Giant step factor: $3^{-m} = 3^{-4} \equiv (3^4)^{-1} = 81^{-1} \equiv 13^{-1} \pmod{17}$. Since $13 \times 4 = 52 \equiv 1$, $3^{-4} \equiv 4$.

Giant steps: $13 \times 4^0 = 13$, $13 \times 4^1 = 52 \equiv 1$. Match! 1 appears at baby step $j = 0$. So $x = 1 \times 4 + 0 = 4$. Check: $3^4 = 81 \equiv 81 - 4 \times 17 = 13$. ✓

5.3. Baby-step giant-step stores a table of $\sqrt{n}$ pairs (j, g^j). For $n = 2^{256}$, this is $\sqrt{n} = 2^{128}$ entries. At 32 bytes each, this requires $\sim 2^{128} \times 32 \approx 10^{39}$ bytes of storage—far exceeding all storage on Earth ($\sim 10^{23}$ bytes).

5.4. Pollard's rho generates a pseudo-random walk on the group. By the birthday paradox, after $O(\sqrt{n})$ steps, two values collide with high probability. This is because the probability of no collision after k steps is approximately $\prod_{i=0}^{k-1}(1 - i/n) \approx e^{-k^2/(2n)}$, which drops below $1/2$ when $k \approx 1.2\sqrt{n}$. Floyd's cycle-detection finds the collision in $O(1)$ space.

5.5. Index calculus works in $\mathbb{F}_p^*$ by exploiting the *smoothness* of integers: many integers factor into small primes. The algorithm builds relations $g^{x_i} = \prod p_j^{a_{ij}}$ using a factor base of small primes, then solves a linear system for discrete logs of the factor base elements. For elliptic curves, points don't have a notion of "factoring into small points"—there is no known way to decompose a point into a sum of "small" points from a factor base. This absence of smoothness is precisely why the ECDLP is harder.

Chapter 6 Solutions

6.1. $2^3 + 7 = 15$ and $(\sqrt{15})^2 = 15$. ✓

6.2. (a) $4(3)^3 + 27(5)^2 = 108 + 675 = 783 \neq 0$. Valid. (b) $4(-3)^3 + 27(2)^2 = -108 + 108 = 0$. **Not valid**—the polynomial $x^3 - 3x + 2 = (x-1)^2(x+2)$ has a repeated root. (c) $4(0)^3 + 27(7)^2 = 1323 \neq 0$. Valid.

6.3. Verify: $1^3 - 7(1) + 10 = 4 = 2^2$ ✓; $3^3 - 7(3) + 10 = 16 = 4^2$ ✓.
$\lambda = (4-2)/(3-1) = 1$. $x_3 = 1 - 1 - 3 = -3$. $y_3 = 1(1-(-3)) - 2 = 2$. $P + Q = (-3, 2)$.
Verify: $(-3)^3 - 7(-3) + 10 = -27 + 21 + 10 = 4 = 2^2$. ✓

6.4. $P = (1, \sqrt{8})$ on $y^2 = x^3 + 7$. Doubling with $a = 0$: $\lambda = 3(1)^2/(2\sqrt{8}) = 3/(2\sqrt{8}) = 3\sqrt{8}/16 = 3\sqrt{2}/8 \approx 0.5303$.
$x_3 = \lambda^2 - 2 = 9/32 - 2 = -55/32 \approx -1.7188$.
$y_3 = \lambda(1 - x_3) - \sqrt{8} = (3\sqrt{2}/8)(1 + 55/32) - 2\sqrt{2} = (3\sqrt{2}/8)(87/32) - 2\sqrt{2} = 261\sqrt{2}/256 - 2\sqrt{2} = (261 - 512)\sqrt{2}/256 = -251\sqrt{2}/256$.
So $2P = (-55/32, -251\sqrt{2}/256)$.

6.5. The line through P and Q is the same geometric object as the line through Q and P—it intersects the curve at the same third point R' regardless of the order, so reflecting gives the same R.

6.6. Substitute $y = \lambda(x - x_1) + y_1$ into $y^2 = x^3 + ax + b$: $[\lambda(x - x_1) + y_1]^2 = x^3 + ax + b$. Expanding: $\lambda^2(x - x_1)^2 + 2\lambda y_1(x - x_1) + y_1^2 = x^3 + ax + b$. Since P is on the curve, $y_1^2 = x_1^3 + ax_1 + b$. Rearranging: $x^3 - \lambda^2 x^2 + (\text{lower terms}) = 0$. The three roots are x_1, x_2, x_3. By Vieta's formulas, $x_1 + x_2 + x_3 = \lambda^2$, giving $x_3 = \lambda^2 - x_1 - x_2$.

Chapter 7 Solutions

7.1. By Euler's criterion, c is a QR mod 7 iff $c^3 \equiv 1 \pmod 7$.

$1^3 = 1 \equiv 1$ (QR). $2^3 = 8 \equiv 1$ (QR). $3^3 = 27 \equiv 6 \equiv -1$ (NR). $4^3 = 64 \equiv 1$ (QR). $5^3 = 125 \equiv 6 \equiv -1$ (NR). $6^3 = 216 \equiv 6 \equiv -1$ (NR).

QRs: $\{1, 2, 4\}$. These are $1^2 = 1, 3^2 = 2, 2^2 = 4 \pmod 7$. ✓

7.2. $E : y^2 = x^3 + 2x + 3$ over $\mathbb{F}_7$. For each x: $x = 0$: $c = 3$, $3^3 = 6 \equiv -1$, NR. $x = 1$: $c = 6$, $6^3 = 216 \equiv 6 \equiv -1$, NR. $x = 2$: $c = 15 \equiv 1$, QR, $y = \pm 1$, pts $(2, 1), (2, 6)$. $x = 3$: $c = 36 \equiv 1$, QR, $y = \pm 1$, pts $(3, 1), (3, 6)$. $x = 4$: $c = 75 \equiv 5$, $5^3 = 125 \equiv 6 \equiv -1$, NR. $x = 5$: $c = 138 \equiv 5$, NR. $x = 6$: $c = 231 \equiv 0$, $y = 0$, pt $(6, 0)$.

Points: $(2, 1), (2, 6), (3, 1), (3, 6), (6, 0), \mathcal{O}$. Total: $\#E = 6$.

Hasse: $|8 - 6| = 2 \leq 2\sqrt{7} \approx 5.29$. ✓

7.3. $P = (1, 10)$ on $E : y^2 = x^3 + 7$ over $\mathbb{F}_{23}$. From the text, $2P = (22, 11)$.

Now $3P = 2P + P = (22, 11) + (1, 10)$.

$\lambda = (11 - 10)/(22 - 1) = 1/21 \pmod{23}$. 21^{-1}: $21 \times 11 = 231 = 10 \times 23 + 1$, so $21^{-1} = 11$. $\lambda = 11$.

$x_3 = 11^2 - 22 - 1 = 121 - 23 = 98 \equiv 98 - 4 \times 23 = 6$.

$y_3 = 11(22 - 6) - 11 = 11 \times 16 - 11 = 176 - 11 = 165 \equiv 165 - 7 \times 23 = 4$.

$3P = (6, 4)$. Verify: $6^3 + 7 = 223 \equiv 223 - 9 \times 23 = 16$ and $4^2 = 16$. ✓

7.4. When $p \equiv 3 \pmod 4$, $(p + 1)/4$ is an integer. If c is a QR with $c = y^2$, then $c^{(p+1)/4} = y^{(p+1)/2} = y \cdot y^{(p-1)/2} = y \cdot 1 = y$ (using Fermat). For $p = 23$: $\sqrt{8} = 8^{(23+1)/4} = 8^6 \pmod{23}$. $8^2 = 64 \equiv 18$. $8^4 \equiv 18^2 = 324 \equiv 324 - 14 \times 23 = 2$. $8^6 = 8^4 \times 8^2 \equiv 2 \times 18 = 36 \equiv 13$. Check: $13^2 = 169 \equiv 169 - 7 \times 23 = 8$. ✓ (The other root is $-13 \equiv 10$.)

7.5. Hasse gives $|p + 1 - \#E| \leq 2\sqrt{p}$, so the interval width is $4\sqrt{p}$. For $p \approx 2^{256}$: width $\approx 4 \times 2^{128} = 2^{130}$. As a fraction of p: $2^{130}/2^{256} = 2^{-126} \approx 10^{-38}$. The uncertainty in the group size is negligibly small relative to p.

Chapter 8 Solutions

8.1. sec = Standards for Efficient Cryptography; p = prime field; 256 = 256-bit prime; k = Koblitz (efficiency-optimized parameters); 1 = first such curve.

8.2. (1) $p = 2^{256} - 2^{32} - 977$: the field prime. (2) $a = 0$: curve coefficient (Koblitz). (3) $b = 7$: curve coefficient ($y^2 = x^3 + 7$). (4) G: generator point (base point). (5) n: group order (prime). (6) $h = 1$: cofactor.

8.3. $2^{256} \bmod 4 = 0$. $2^{32} \bmod 4 = 0$. $977 \bmod 4 = 1$ (since $977 = 244 \times 4 + 1$). So $p \equiv 0 - 0 - 1 = -1 \equiv 3 \pmod 4$. ✓

8.4. With $h = 1$, $\#E(\mathbb{F}_p) = n$ (prime), so there are no nontrivial subgroups (by Lagrange's theorem). Every non-identity point has order n. With $h = 8$ (Curve25519), there is a subgroup of order 8. An attacker could send a point of order 8, and the result of scalar multiplication would leak the private key modulo 8. Defense: multiply received points by 8 ("cofactor multiplication") to project into the prime-order subgroup, or validate that the point has the correct order.

8.5. In Dual_EC_DRBG, the standard specifies two points P and Q on an elliptic curve. If $Q = eP$ for some secret e, and the DRBG outputs $r_i = x(s_iP)$ where $s_{i+1} = x(s_iP)$, then someone knowing e can compute $s_iQ = s_i(eP) = e(s_iP)$, recovering the internal state. The NSA chose Q without publishing the relationship to P.

secp256k1 avoids this: G was chosen as the point with the smallest x-coordinate generating the correct order, a deterministic "nothing up my sleeve" construction. Anyone can verify this choice independently.

Chapter 9 Solutions

9.1. At most 256 doublings and 256 additions = 512 EC operations. Naive repeated addition: $d \leq 2^{256}$ additions. Ratio: $2^{256}/512 \approx 2^{247}$—double-and-add is astronomically faster.

9.2. $100 = (1100100)_2$. Trace:

Bit 1: $R = \mathcal{O} \to 2\mathcal{O} = \mathcal{O} \to \mathcal{O} + P = P$. Bit 1: $R = 2P \to 2P + P = 3P$. Bit 0: $R = 6P$. Bit 0: $R = 12P$. Bit 1: $R = 24P \to 25P$. Bit 0: $R = 50P$. Bit 0: $R = 100P$. ✓

7 doublings + 3 additions = 10 operations.

9.3. Given prefix byte and x-coordinate: (1) Compute $c = x^3 + 7 \bmod p$. (2) Compute $y = c^{(p+1)/4} \bmod p$ (valid since $p \equiv 3 \pmod 4$). (3) If y is even and prefix is 02, or y is odd and prefix is 03, output (x, y). Otherwise output $(x, p - y)$. The condition $p \equiv 3 \pmod 4$ makes step 2 a single exponentiation rather than the more complex Tonelli–Shanks.

9.4. Not immediately. A Bitcoin address is RIPEMD-160(SHA-256(Q)). To steal funds, an attacker needs Q (to solve the ECDLP), but Q is only revealed when coins are *spent* (in the spending transaction's input). Before spending, only the address hash is public. A quantum attacker would need to break RIPEMD-160 (find a preimage) to recover Q from the address—a different (and potentially harder) problem. However, reused addresses and pay-to-pubkey outputs expose Q directly.

9.5. $2^{128}/(10^{15} \times 3.156 \times 10^7) = 2^{128}/(3.156 \times 10^{22})$. $2^{128} \approx 3.4 \times 10^{38}$. So the time is $\approx 3.4 \times 10^{38}/(3.156 \times 10^{22}) \approx 1.08 \times 10^{16}$ years $\approx 10^{16}$ years. (The age of the universe is $\sim 10^{10}$ years, so this is $\sim 10^6$ times the age of the universe.)

Chapter 10 Solutions

10.1. The two components are r (the x-coordinate of kG reduced mod n) and s (the signing equation value). Each is 32 bytes, totaling 64 bytes.

10.2. All arithmetic mod $n = 101$. From nonce reuse: $k = (z_1 - z_2)(s_1 - s_2)^{-1} = (42 - 99)(17 - 53)^{-1} = (-57)(-36)^{-1} \equiv 44 \times (-36)^{-1} \pmod{101}$.

Find $(-36)^{-1} \equiv (65)^{-1} \pmod{101}$: $65 \times 14 = 910 = 9 \times 101 + 1$, so $65^{-1} = 14$.

$k = 44 \times 14 = 616 \equiv 616 - 6 \times 101 = 10$.

Now $d = r^{-1}(s_1 k - z_1) = 37^{-1}(17 \times 10 - 42) = 37^{-1}(128) \pmod{101}$.

$128 \equiv 27$. 37^{-1}: $37 \times 30 = 1110 = 10 \times 101 + 100 \equiv -1$, so $37^{-1} = -30 \equiv 71$.

$d = 71 \times 27 = 1917 \equiv 1917 - 18 \times 101 = 1917 - 1818 = 99$.

10.3. $r = x_{kG} \bmod n$. If two signatures have the same r, the nonce points $R_1 = k_1 G$ and $R_2 = k_2 G$ have the same x-coordinate mod n. The overwhelming probability is $k_1 = k_2$ (exact reuse). This immediately enables key recovery via the nonce reuse theorem.

10.4. Verification computes $w = s^{-1}$, then checks $x_P = r$ where $P = zwG + rwQ$. With $s' = n - s$, we get $w' = (n - s)^{-1} = -s^{-1} = -w$. Then $P' = (-zw)G + (-rw)Q = -(zwG + rwQ) = -P$. Since $-P = (x_P, -y_P)$, the x-coordinate is unchanged: $x_{P'} = x_P = r$. ✓

BIP 62 requires $s \leq n/2$ to select a unique canonical representative, since exactly one of s and $n - s$ satisfies this.

10.5. RFC 6979 is secure because HMAC-SHA256 is a PRF (pseudorandom function). Without d, an attacker cannot compute $k = \text{HMAC}(d, z)$, so k looks random. Two signatures produce the same k only if the same private key signs the same message hash—which is fine, as it produces the identical signature (no information leak).

Chapter 11 Solutions

11.1. $sG = R + eQ$. This requires 2 scalar multiplications (sG and eQ) plus one point addition. (In practice, multi-scalar multiplication algorithms like Strauss/Shamir can compute $sG + (-e)Q$ in roughly 1.5 scalar multiplications.)

11.2. An x-only public key encodes just the x-coordinate (32 bytes), with y implicitly chosen to be even. A compressed public key is 33 bytes (1-byte prefix + 32-byte x). Savings: 1 byte per public key.

11.3. $sG = (k + ed)G = kG + edG = R + e(dG) = R + eQ$. ✓

11.4. Instead of m individual checks $s_iG = R_i + e_iQ_i$, we check $(\sum \alpha_i s_i)G = \sum \alpha_i R_i + \sum \alpha_i e_i Q_i$. The left side is one scalar multiplication; the right side is a multi-scalar multiplication of $2m$ points, computable in roughly $m + 1$ scalar multiplications via Pippenger's algorithm. Total: $\approx m + 2$ scalar multiplications vs. $2m$ individually. For large m, this approaches 2× speedup. The random weights α_i ensure soundness: if any individual equation fails, the batch equation fails with probability $\geq 1 - 1/n$ (essentially certain).

11.5. Schnorr's $s = k + ed$ is linear in d and k (no products of secrets, no inversions). ECDSA's $s = k^{-1}(z + rd)$ involves multiplying k^{-1} by rd—a nonlinear combination. Linearity enables: (1) Signature aggregation: partial signatures $s_i = k_i + ed_i$ sum to $\tilde{s} = \tilde{k} + e\tilde{d}$, which is valid for the aggregate key. (2) Batch verification: linear equations combine cleanly.

11.6. In BIP 340, R is encoded as its x-coordinate with y implicitly even (if y_R was odd, k was negated during signing). Given (R, s), the values R, $e = H(R\|Q\|m)$, and the equation $sG = R + eQ$ uniquely determine s (since the equation has a unique solution for s given R and e). There is no second valid s.

In ECDSA, $r = x_R$ discards y_R's sign, and the equation $s = k^{-1}(z + rd)$ doesn't pin down the sign of the verification point—leading to both (r, s) and $(r, n - s)$ being valid.

Chapter 12 Solutions

12.1. In naive aggregation ($\tilde{Q} = Q_A + Q_B$), an adversary Mallory with true key Q_M announces $Q'_M = Q_M - Q_A$ as her public key. The aggregate becomes $\tilde{Q} = Q_A + Q'_M = Q_A + Q_M - Q_A = Q_M$. Mallory controls the aggregate key and can sign alone, completely excluding Alice.

12.2. The coefficients $a_i = H(L, Q_i)$ depend on the entire list $L = (Q_1, \ldots, Q_m)$ of public keys. If Mallory changes her announced key to Q'_M, the list L changes, which changes *every* coefficient a_i, including a_A. There is no way for Mallory to choose Q'_M such that $a_A Q_A + a_M Q'_M = Q_M$ because the hash function makes a_A and a_M unpredictable functions of Q'_M.

12.3. MuSig1 has 3 rounds: (1) commit to nonces, (2) reveal nonces, (3) send partial signatures. MuSig2 has 2 rounds: (1) share nonce pairs, (2) send partial signatures. The commitment round in MuSig1 prevents **Wagner's attack**: if an adversary sees all nonces before choosing their own, they can solve a generalized birthday problem to find a nonce that forges a signature. MuSig2 eliminates this round by using multiple nonces combined with a message-dependent binding factor.

12.4. The binding factor b determines how the two nonces are combined: $R_i = R_{i,1} + b \cdot R_{i,2}$. If b were independent of m, an adversary who received pre-shared nonces (Round 1) could mount Wagner's attack by choosing m to make the combined nonce vulnerable. By including m in b's hash, the adversary cannot control b before committing to a message, defeating the attack.

12.5. We need $\tilde{s}G = \tilde{R} + e\tilde{Q}$.
$\tilde{s} = \sum_i s_i = \sum_i [(k_{i,1} + b \cdot k_{i,2}) + e \cdot a_i \cdot d_i]$.
$\tilde{s}G = \sum_i (k_{i,1} + b \cdot k_{i,2})G + e \sum_i a_i d_i G = \sum_i (R_{i,1} + b \cdot R_{i,2}) + e \sum_i a_i Q_i = \sum_i R_i + e\tilde{Q} = \tilde{R} + e\tilde{Q}$. ✓

12.6. MuSig2 is m-of-m: all signers must participate. FROST enables t-of-m ($t \leq m$): any t signers can produce a valid signature. The threshold case requires **Shamir's secret sharing**: the private key is split into m shares using a degree-$(t-1)$ polynomial, and any t shares can reconstruct the key via **Lagrange interpolation**. Each signer holds a share $d_i = f(i)$ where f is a polynomial with $f(0) = d$. During signing, each participant i computes a partial signature weighted by their Lagrange coefficient $\lambda_i = \prod_{j \neq i} j/(j-i)$. The additional complexity: (1) a distributed key generation (DKG) protocol replaces simple key generation; (2) Lagrange coefficients must be computed and verified; (3) the security proof must handle a corrupted subset of size $< t$.