Lorenzo Naranjo November 11, 2026

Hashing and Proof of Work

A payment system needs a ledger: a record of who owns what. In a bank, the bank keeps the ledger and everyone trusts it. Bitcoin keeps a ledger with no one in charge, copied on thousands of computers run by people who do not know or trust each other. This chapter explains the two ideas that make that possible, hash functions and proof of work, and why the result is expensive to cheat.

The problem: double spending

Digital money is just data, and data can be copied. If Alice can send the same coin to both Bob and Carol, the money is worthless. A bank prevents this *double spending* by checking every payment against its ledger. Without a bank, the copies of the ledger have to agree on which payments happened and in what order, even though anyone can join the network and some participants may try to cheat. Nakamoto (2008) proposed the solution Bitcoin uses: put payments in a public chain of blocks, and make adding a block costly.

Hash functions

A *hash function* takes any input, a word, a file, or a block of payments, and produces a short output of fixed length, called its hash. Bitcoin uses SHA-256, whose output is 256 bits, usually written as 64 characters in hexadecimal, the base-16 number system with digits 0 to 9 and a to f.[1]

Think of a hash as a fingerprint of the input. The hash of “hello world” is

b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9

and the hash of “Hello World”, with two capital letters, is

[1] A bit is a 0 or a 1, and a byte is 8 bits, enough to store one character of text. One hexadecimal digit stands for 4 bits, since $2^4 = 16$, so 256 bits take $256/4 = 64$ hexadecimal characters. Nothing in this chapter depends on these details; they explain why hashes look the way they do.

a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e

Four properties make hash functions useful:

- **The same input always gives the same hash.** Anyone can recompute it and check it.
- **A tiny change in the input changes the hash completely,** as the two examples show.
- **The hash cannot be reversed.** Knowing a hash tells you nothing about the input that produced it, except by trying inputs one at a time.
- **No one can find two inputs with the same hash.** Such pairs must exist, since there are infinitely many possible inputs, but there are 2^{256} possible hashes, a number with 78 digits, and no one has ever found two inputs that share a SHA-256 hash.

Hashes are used far beyond cryptocurrencies. Software sites publish the hash of each installer, so you can check that the file you downloaded is the one they published and was not altered on the way.

Blocks and the chain

Bitcoin groups payments into *blocks*. Each block has a header that contains, among other things, a single hash summarizing all the payments in the block and the hash of the previous block's header. Because each block contains the hash of the one before it, the blocks form a *chain*.

The chain makes the past tamper-evident. Suppose someone changes a payment in an old block. The payments no longer match the summary in the block's header, so anyone who checks sees that the block is invalid. If the cheater also updates the summary, the header and its hash change, and the next block, which stored the old hash, no longer points to it. Either way the chain is broken at the altered block, and anyone checking sees it immediately. The proof of work notebook builds a small chain and shows exactly this.

Tamper-evident is not tamper-proof: a cheater could change a block and then repair the chain, recomputing the altered block's hash, then the next block's, which must now store the new hash, and so on to the end. Proof of work makes each of those recomputations expensive.

Mining: a lottery run on computation

To add a block, a *miner* must find a header whose hash is below a threshold called the *target*. A hash is a number, and a hash that starts with zeros is a small one, so in the simplified version used in these notes the target says that the hash must start with a given number of zeros. Bitcoin's actual target can be any number, so its difficulty moves in small steps rather than in factors of 16. The miner cannot compute the right header directly, because hashes cannot be reversed. All it can do is change a number in the header called the *nonce*, hash again, and check, over and over.

Each hexadecimal digit of a hash is equally likely to be any of 16 values, and the digits behave like independent draws. So the chance that a hash starts with one zero is $1/16$, with two zeros $1/16^2 = 1/256$, and with N zeros

$$s = \frac{1}{16^N}. \tag{1}$$

Each try is a draw in a lottery with success probability s, so the expected number of tries until the first success is

$$\mathrm{E}[\text{tries}] = \frac{1}{s} = 16^N. \tag{2}$$

Every additional zero multiplies the expected work by 16. Bitcoin adjusts its target every 2,016 blocks so that, whatever the total computing power on the network, a block is found about every ten minutes. When more miners join, the target falls and each block takes more tries.

Mining is a race. Every miner is running the same lottery, and the first to find a valid hash broadcasts the block and collects the reward: newly created bitcoin plus the fees attached to the payments in the block. A miner with 10% of the network's computing power wins about 10% of the blocks. Because a single miner may wait months for a win, most miners join *pools* that combine their computing power and share the rewards, which turns a lumpy income into a steady one.

The other side of the race matters just as much: checking a block is cheap. Producing a valid block takes on the order of 16^N hashes, but anyone can verify its proof of work

with one, and checking its payments takes little more. Production is expensive and verification is nearly free, so every participant can check every block.

Why rewriting the past is expensive

Once a payment is in a block, miners build new blocks on top of it. Wallets call these blocks *confirmations*. To reverse a payment, an attacker must build an alternative chain, without the payment, that becomes longer than the honest one, while honest miners keep extending theirs. The network follows the longest chain, the one with the most work behind it.

Let q be the attacker's share of the network's computing power and $p = 1 - q$ the honest miners' share. Nakamoto (2008) first asks a simpler question: if the attacker is z blocks behind, what is the probability that they ever catch up? The answer is

$$Q_z = \begin{cases} 1, & \text{if } p \leq q, \\ (q/p)^z, & \text{if } p > q. \end{cases} \tag{3}$$

An attacker with half the computing power or more eventually wins. An attacker with less faces odds that fall exponentially with how far behind they are. The formula assumes the attacker can keep trying forever, with unlimited funds.

A real attacker does better than (3) suggests, because they do not start from scratch when the merchant has seen z blocks on top of the payment. They start mining their alternative chain in secret the moment they send the payment, so by the time the merchant has seen z blocks, the attacker has usually built some blocks of their own and is fewer than z blocks behind. Nakamoto accounts for this head start and computes the probability that the attacker eventually succeeds. Figure 1 shows it, and his paper tabulates how many blocks a merchant must wait for to bring the probability below 0.1%: 5 when the attacker has 10% of the computing power, 15 at 25%, 24 at 30%, and 89 at 40%.

This is why merchants wait for several blocks before treating a large payment as final. How many is a business decision: an attacker gains more from reversing a large payment, so it makes sense to wait longer before treating one as final.

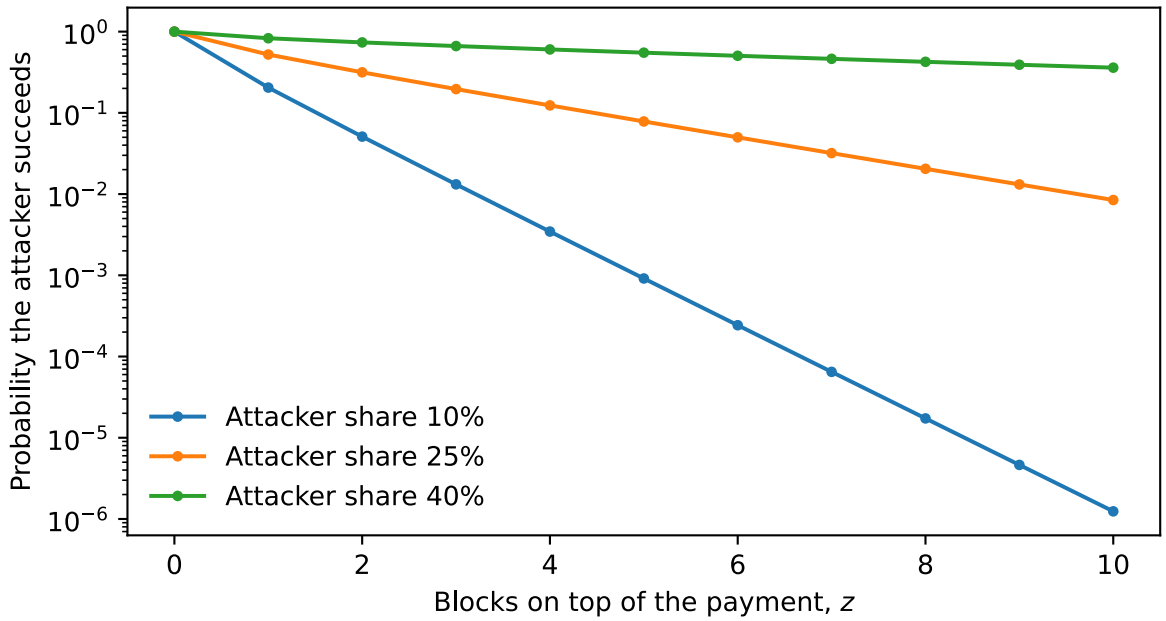


Figure 1: Probability that an attacker with share q of the computing power reverses a payment after the merchant has seen z blocks on top of it, allowing for the blocks the attacker mined in secret in the meantime, from the calculation in Nakamoto (2008).

Who pays for security

Proof of work keeps the ledger honest by making attacks expensive, and someone has to pay for that expense. Miners spend on hardware and electricity, and they do so only because the block rewards and fees pay them for it. Budish (2025) draws the economic conclusion: the payments to miners must be large relative to what an attacker could gain by attacking. The security of the chain is therefore a recurring cost, paid in every block. As the newly created bitcoin in each block halves roughly every four years, either the price of bitcoin or the fees users pay must make up the difference, or the budget for security shrinks.

Practice Problems

These problems give you a chance to practice the concepts introduced in this chapter. Try to solve each one on your own before expanding the solution.

Solutions to all problems can be found at https://lorenzonaranjo.com/class-materials/fintech/.

Problem 1 (How Long Does Mining Take?). A blockchain requires block hashes to start with 6 zeros in hexadecimal. A single computer computes 2 million hashes per second. On average, how many tries does it take to find a valid block, and how long does that take on this computer?

Problem 2 (Tampering with an Old Block). A chain has 10 blocks. Someone changes one payment in block 3 and leaves everything else as it was. Which checks fail, and what would the cheater have to do to make the change stick?

Problem 3 (How Many Blocks?). An attacker controls 30% of the network's computing power. Using (3), what is the probability that they ever catch up if they are 6 blocks behind? A merchant waits for 6 blocks on top of a payment from this attacker; why is the merchant's risk higher than your answer, and what does Nakamoto's table say about how many blocks the merchant should wait for to keep the risk below 0.1%?

References

Budish, Eric. 2025. "Trust at Scale: The Economic Limits of Cryptocurrencies and Blockchains." *Quarterly Journal of Economics* 140 (1): 1–62. https://doi.org/10.1093/qje/qjae033.

Nakamoto, Satoshi. 2008. "Bitcoin: A Peer-to-Peer Electronic Cash System.". https://bitcoin.org/bitcoin.pdf.