Mastering Zcash

Maxime Desalle

2025-01-12

Figure 1: Leonard Bernstein's "Ode to Freedom" concert on Christmas day in 1989 celebrating the fall of the Berlin Wall. The orchestra consisted of members representing the two German States and the four occupying powers of post-war Berlin. The concert was broadcast live to an estimated audience of 100 million people in more than twenty countries. The victory of freedom, democracy, and capitalism, over oppression, totalitarianism, and communism — pictured.

With deep gratitude to Giulia Mouland for her feedback and editorial review, and to Arjun Khemani for his support.

---

1. Introduction

Unless you're using cash, the information about every purchase that you make is tracked and stored indefinitely. It doesn't matter what it is, or how sensitive it is. The infrastructure that powers commerce, both offline and online, has effectively become an inescapable surveillance apparatus.

When it was first released, there were hopes that Bitcoin could fix this, but unfortunately, it hasn't. In fact, contrary to many people's understanding, Bitcoin is incredibly transparent, as every transaction ever made is permanently stored and visible to everyone. Sure, wallets are pseudonymous, but in order to receive BTC you need to provide your address, thus providing your entire transaction history and balance to the sender. On top of that, services like Arkham have made it trivial, for even the general public, to track and identify wallets.

This is why authorities condone Bitcoin, for to state actors, transparent chains are better than the digital currencies that they themselves control (often called *Central Bank Digital Currencies* or *CBDCs*) in many ways. Since there is no resistance from the population to using Bitcoin, and no oversight on how chain data is used by authorities, it offers perfect visibility for state actors to track everything, with full impunity.

In some ways, Bitcoin is actually worse than the banking system it sought to replace. At least bank records are private from the general public; Bitcoin isn't.

It's for this reason that Zcash takes a different approach: offering default privacy, rather than default transparency. This means that when you make a *shielded* Zcash transaction, the sender, the recipient, and the transaction amount are all encrypted. The network verifies the transaction is valid, verifying that you have the funds and aren't spending more ZEC than you own, but isn't privy to any information about the transaction itself.

> **Note** ZEC is the symbol or ticker for Zcash, like what BTC is for Bitcoin.

Initially, when you think about it, this sounds impossible. How can you prove that something is true without revealing the thing that you're proving? The answer is zero-knowledge proofs, specifically a construction called *zk-SNARKs*. The coverage of zk-SNARKs in this article will be kept light and accessible to the general reader, as it requires a substantial background in algebra and commitment schemes—beyond this article's scope.

We will also cover Zcash's origins in academic cryptography, the philosophy that shaped it, and the protocol as it exists today.

Some parts of this comprehensive study of Zcash will be more technical. Though I have tried to make things as clear and accessible as possible for everyone, if you have trouble with certain concepts, I recommend asking an LLM for clarification or simply skipping it and revisiting it later. If that doesn't work, don't hesitate to reach out with any questions.

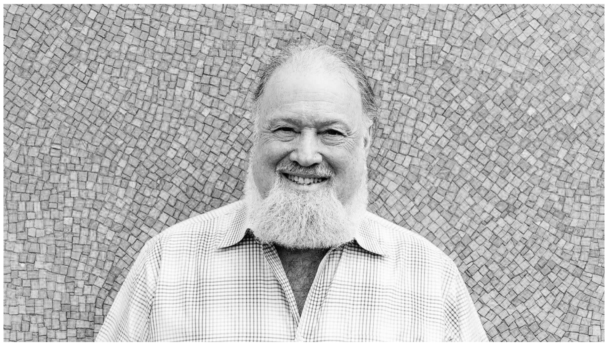

Figure 2: David Chaum, cryptography pioneer.

2. Origins

2.1 David Chaum and the Birth of Digital Cash

The idea of private digital money is far from new, in fact, it dates back to 1982. David Chaum, who was then a PhD candidate in computer science, published a paper titled *"Blind Signatures for Untraceable Payments."*

The core insight of this paper was simple and elegant: a bank could sign a digital token without seeing its content, just as you could sign the outside of a sealed envelope. Then, when the token was spent, the bank could verify its validity through its own signature, but wouldn't be able to link the spending to the withdrawal.

Later, in 1989, David Chaum founded DigiCash, a company built to commercialize this idea. The product was called ecash and it enabled users to withdraw digital tokens from their bank accounts and spend them at merchants without leaving a trail connecting the buyer to the purchase. Several banks piloted the technology, including Deutsche Bank and Credit Suisse.

Unfortunately, DigiCash didn't succeed, the timing was wrong. Recall that this was created before widespread internet commerce, and before people understood the importance of online privacy. The company filed for bankruptcy in 1998, but with ecash, Chaum had proven that private digital money was doable.

2.2 The Cypherpunks

Soon after, a different kind of movement started taking shape. In 1992, a group of cryptographers, hackers, and libertarians started meeting in the San Francisco Bay Area and communicating via an electronic mailing list. They called themselves the *cypherpunks*.

The cypherpunks were not academics writing papers, they were ideologues writing code. Their founding premise was that in the digital age, privacy would not be granted by governments or corporations, instead, it would have to be built, deployed, and defended by individuals using cryptographic tools. In 1993, group member Eric Hughes crystallized this concept in *A Cypherpunk's Manifesto*:

"Privacy is necessary for an open society in the electronic age... We cannot expect governments, corporations, or other large, faceless organizations to grant us privacy out of their beneficence... We must defend our own privacy if we expect to have any... Cypherpunks write code."

The mailing list became a crucible for the ideas that would shape the next three decades of cryptographic development. Members included Julian Assange (before WikiLeaks), Hal Finney (who would later receive the first Bitcoin transaction), Nick Szabo (who proposed *bit gold*, a conceptual precursor to Bitcoin), and Wei Dai (whose *b-money* proposal was cited by Satoshi Nakamoto). In 1997, another member, Adam Back, invented *Hashcash*, the *Proof of Work (PoW)* system later adopted by Bitcoin.

The cypherpunks didn't build a successful private currency, or did they? The creation of Bitcoin is attributed to the pseudonymous Satoshi Nakamoto, rumoured to have been a developer or a group of developers tied to the cypherpunks, and who has not been active in over a decade. In any case, what we know for sure, is that the cypherpunks built the culture, the tools, and the intellectual framework that has made private currency possible.

2.3 Bitcoin: The Wrong Tradeoff

On October 31, 2008, Satoshi Nakamoto posted a paper to a cryptography mailing list titled *"Bitcoin: A Peer-to-Peer Electronic Cash System."* The paper described a solution to a problem that had plagued digital currency designers for decades: how do you prevent double-spending without relying on a central authority?

Satoshi's proposed answer was the blockchain: a public ledger maintained by a decentralized network of miners, secured by PoW; it was brilliant, and it worked! Bitcoin launched in January of 2009, and for the first time, people could transfer value over the internet without banks, intermediaries, or permission.

> **Note** We will cover what miners and Proof of Work (PoW) are and how they work in the context of Zcash later in this article.

However, there was one glaring problem, as mentioned above, Bitcoin isn't private. The blockchain is entirely public by design: every transaction, every address, and every balance are visible to anyone who's interested. Satoshi acknowledged this problem in the paper, suggesting that users could preserve some of their privacy by using new addresses for each transaction, but this was weak mitigation, as addresses can be clustered, transaction graphs can be analyzed and real-world identities can be linked through exchanges, merchants, and metadata.

Nakamoto also later acknowledged that a privacy-preserving form of Bitcoin would enable a cleaner implementation of the protocol, but at the time, he couldn't envision how to bring it about with zero-knowledge proofs.

Problematically, the privacy problem remained overlooked for years. Early Bitcoin users assumed pseudonymity was close enough to anonymity, but they were wrong. By the early 2010s, researchers demonstrated that blockchain analysis could deanonymize users with high accuracy. Companies like Chainalysis, founded in 2014, turned this into a business by selling blockchain forensics to law enforcement agencies, exchanges, and even governments.

Bitcoin had solved the double-spend problem, but it had made the privacy problem worse.

2.4 Zerocoin: The Bolt-On Attempt

In 2013, Matthew Green, a cryptographer at Johns Hopkins University, and two graduate students, Ian Miers and Christina Garman, published *"Zerocoin,"* a paper proposing a solution to Bitcoin's problem.

Their idea was to add a privacy layer on top of Bitcoin, such that users could convert their bitcoins into *zerocoins*, anonymous tokens with no transaction history. Later, when you wanted to spend it, you could convert it back to Bitcoin. The conversion process relied on cryptographic techniques known as zero-knowledge proofs, which let you prove that you owned a valid zerocoin without revealing its origin.

Zerocoin worked in theory, but it had problems. First, the proofs were large, two orders of magnitude larger than the few hundred bytes required for a normal Bitcoin transaction. Second, the cryptography was also limited: you could prove ownership, but you couldn't hide transaction amounts. Third, and most critically, it required Bitcoin to adopt it as a protocol change, but Bitcoin's conservative development culture made that unlikely.

The Bitcoin community debated Zerocoin and ultimately decided to pass on it. The proposal never made it into the protocol.

2.5 Zerocash: The Rebuild

In 2014, a new paper was published. The author list had expanded to include Eli Ben-Sasson and Alessandro Chiesa, cryptographers who had been working on a new generation of zero-knowledge proofs, plus Eran Tromer and Madars Virza.

The paper was titled *"Zerocash: Decentralized Anonymous Payments from Bitcoin."* Despite what its title may lead you to think, it wasn't simply a Bitcoin extension, it was a complete redesign.

The key innovation was the use of zk-SNARKs, which stands for *Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge.* These were zero-knowledge proofs that were small (a few hundred bytes), fast to verify (milliseconds), and expressive enough to prove complex statements about hidden data. With zk-SNARKs you can prove not just that you own a valid coin, but prove that an entire transaction is valid. This isn't trivial, it means that the system verifies that the transaction amounts are correct, there is no double-spending, etc., all without revealing the sender, recipient, or amount.

However, there was a catch: zk-SNARKs required a trusted setup. Someone had to generate a set of public parameters that the system would use forever, but, if that person kept the secret values used to generate the parameters, it's so-called *toxic waste*, they could undetectably create counterfeit coins. Though this was of serious concern, the researchers believed it could be prevented with careful ceremony design.

2.6 The Genesis Block

Zooko Wilcox had been in the privacy and cryptography space for decades. He had worked at DigiCash in the 1990s and been involved with decentralized storage projects with strong privacy properties like Tahoe-LAFS. So, when the Zerocash paper was released, it was an immediate fit.

In 2016, Wilcox founded the *Zcash Company*, later renamed *Electric Coin Company*, and assembled a team to turn Zerocash into a production cryptocurrency. The academic authors mentioned above joined as advisors and collaborators on the project.

The trusted setup problem highlighted above required a creative solution. The team designed an elaborate, multi-party computation ceremony: six participants, all in different locations around the world, would contribute randomness to generate the public parameters, and as long as at least one participant destroyed their secret input, the toxic waste would be unrecoverable. The ceremony took place in late 2016, with participants including Peter Todd, a Bitcoin Core developer, and journalists who documented the process. Extensive work went into making sure that the ceremony wasn't compromised, as outlined here.

On October 28, 2016, the Zcash genesis block was mined. For the first time, a production cryptocurrency offered genuine, cryptographic privacy. Thirty-four years after David Chaum's first paper, the dream of untraceable digital money was running on a live network.

Figure 3: Hyperinflation in the Weimar Republic. Banknotes had lost so much value that they were used as wallpaper.

3. What is Zcash?

3.1 A Bitcoin Primer

> **Tip** If you already understand how Bitcoin works, feel free to skip ahead, this section is for readers unfamiliar with Bitcoin's inner workings.

Bitcoin is essentially a payment system with no central operator. There is no bank, no company, and no single server that can be pointed to. Its decentralized mechanism operates through thousands of computers around the world that maintain identical copies of a shared ledger, called the *blockchain*, and follow a set of rules to keep it in sync.

The blockchain is an append-only data structure, and it's literally a chain of blocks, so you can add new entries (blocks), but you can never modify or delete old ones. Each new block consists of transactions made on the network at the time the block was created. Additionally, each block references the one preceding it, leading to the formation of a chain. If you wanted to change a transaction from the past, you'd have to rewrite every successive block, which becomes computational impossibility once enough time has passed. We will see why that is the case later.

Keys and Ownership Bitcoin uses public-key cryptography for wallets. When you "create a wallet," what you're really doing is generating a key pair: a private key (a large random number, kept secret) and a corresponding public key (derived mathematically from the private key). A Bitcoin address is derived from a public key through hashing and encoding.

> **Example** Here's an example of what these look like in practice (abbreviated using ...):
>
> - Private key: `1E99423A4ED27608A15...E6E9F3A1C2B4D5F6A7B8C9D0`
> - Public key: `03F028892BAD7ED57D2F...3A6A6C6E7F8C9D0A1B2C3D4E5F607182`
> - Bitcoin address: `1BoatSLRHtKNngkdXEeobR76b53LETtpyT`

The private key lets you sign messages, while the public key lets anyone verify that a signature came from the corresponding private key without revealing the private key itself. This cryptography is what retains the private key's privacy, as you can sign a message authorizing a transfer using your private key, and the network can verify your signature using your public key, without ever seeing your private key.

An important conclusion here is that this means wallets don't "hold" BTC in any meaningful sense. There's no file on your computer containing coins. Rather, the blockchain holds the record of which addresses control which outputs, and your wallet is just a signing tool, it stores your private keys and uses them to authorize transactions. If you lose your private keys, you lose access to your funds; not because the coins disappeared, but because you can no longer prove your ownership.

Transactions and UTXOs Transactions are how Bitcoin value moves. When you send BTC, you're publishing a signed message that effectively says: "I authorize the transfer of these coins to this address," but what exactly are these coins?

Bitcoin doesn't track balances, there aren't database entries somewhere saying "Address X has 3.5 BTC." Instead, Bitcoin uses *Unspent Transaction Outputs*, often abbreviated as *UTXOs*. Every transaction consumes existing outputs and then creates new ones. The outputs you control but haven't yet spent are your UTXOs. This means that your "balance" is just the sum of all of your unspent outputs. There's no running tally of coins, just a collection of discrete chunks you control.

> **Example** Here's a quick example: Imagine that you have a $20 bill and you want to buy a $12 item. Obviously, you can't tear the bill in half, so you hand over the $20 and receive $8 in change.
>
> UTXOs work the same way. If you own a 5 BTC output and want to send someone 3 BTC, you need to consume the entire 5 BTC output and create two new ones from it: 3 BTC for the recipient and 2 BTC that return to you as change. Your original 5 BTC output is now 'spent' and can never be used again.

As a result, a Bitcoin transaction is a data structure contains some metadata as well as: 1. **Inputs:** References to UTXOs you're spending, plus signatures proving you control them 2. **Outputs:** New UTXOs being created, each locked to a recipient's public key

Nodes validate that the inputs exist, haven't been spent yet, and have valid signatures. If everything checks out, the transaction is relayed across the network and waits to be included in a miner's block.

> **Example** Here's what a transaction looks like in practice (hashes and addresses are abbreviated using ...):

```
{
  "txid": "c1b4e693...cbdc5821e3",
  "inputs": [
    {
      "prev_txid":
    "7b1eabe...98a14f3f",
      "output_index": 0,
      "signature":
    "304402204e4...1a8768d1d09",
      "pubkey": "0479be66...ffb10d4b8"
    }
  ],
  "outputs": [
    {
      "amount": 3.0,
      "script": "OP_DUP OP_HASH160
    89...ba OP_EQUALVERIFY
    OP_CHECKSIG"
    },
    {
      "amount": 1.99,
      "script": "OP_DUP OP_HASH160
    12...78 OP_EQUALVERIFY
    OP_CHECKSIG"
    }
  ]
}
```

Each input points to a previous transaction's output by referencing its transaction

ID and index, and each output specifies an amount. The signature proves you control the private key. The 0.01 BTC difference between the input of 5 BTC and outputs of 3BTC + 1.99 BTC, is the transaction fee, claimed by the miner.

Mining and Proof of Work (PoW) Transactions don't confirm themselves. They sit in a waiting area in a node called the mempool (memory pool) until a miner includes them in a block. Mining is the process by which new blocks get added to the chain, and it's designed to be expensive. That's a feature, not a bug, as we will see in a minute.

The problem solved by mining is: in a decentralized network with no central authority, who decides which transactions are valid? Who decides their ordering? If two conflicting transactions appear, say, someone tries to spend the same coins twice, who resolves this conflict?

Bitcoin's solution is: in order to create a valid block, a miner must find a number, called a *nonce*, such that when the block header (containing the previous block's hash, a timestamp, etc.) is combined with this nonce and hashed, the resulting hash is below a certain target value. Since cryptographic hashes are effectively random, there's no way to find a valid nonce except by guessing, so miners guess billions of times per second.

Example For example, think of a block as a page of fixed information with one adjustable number on it (the nonce). Let's assume we start counting the nonce at `0`.

A computer turns the entire page into a single output number called a hash. A hash can be something like `6`, or `03a5b20`, ultimately it's just a number (yes, `03a5b20` is a number, because it equals `3,824,416` in decimal). Remember that the nonce is the only adjustable number on the page, changing only the nonce produces a completely different hash (number) each time.

The network requires the hash to be below a fixed threshold value, and if it isn't, the miner changes the nonce and tries again. Finally, the nonce is accepted when the hash meets the threshold requirement.

For example, imagine a case where the threshold value is `5`. The miner has their page of information and starts with a nonce of `0`. If the computer returns a `6`, which is above `5`, the miner tries again, now `1` as a nonce. If this time the computer returns a `4`, which is below `5`, then `1` is accepted as a nonce!

The target is to adjust 2,016 blocks about every two weeks, maintaining an average block time of ten minutes. If blocks are coming too fast, the target decreases, making the puzzle harder, and if blocks are coming too slow, the target increases. The difficulty adjustment is why Bitcoin's block rate stays stable even as total mining power fluctuates.

Example Here's what a block looks like:


```
{
  "header": {
    "version": 536870912,
    "prev_block_hash":
    "0000000...de0e5c842",
    "merkle_root":
    "8b30c5ba1...1e0d5f8a2c1",
    "timestamp": 1701432000,
    "target": "0000004f2c0...0000000",
    "nonce": 2834917243
  },
  "transactions": [
    {
      "txid": "3a1b9c7e...7e8f9a0b1c",
      "inputs": [{ "coinbase":
    "03a5b20...706f6f6c" }],
      "outputs": [{ "amount": 6.25,
    "script": "OP_HASH160
        f1c3...4c6a8 OP_EQUAL" }]
    },
    { "txid": "c1b4e...5821e3" },
    { "txid": "7d5e8...b5c6d7e" }
  ]
}
```


The header is what gets hashed. Miners increment the nonce repeatedly until `SHA256(SHA256(header)) < target`, meaning until applying the SHA256 hash function twice on the header returns a hash below the target value. The first transaction is always the "coinbase" transaction, which creates new coins and pays the miner.

Once a miner finds a valid nonce, they broadcast the block and other nodes verify it, checking that the hash meets the target, that all transactions are valid, and that the miner didn't create more coins than allowed. If valid, nodes append the block to their chain and begin working on the next one. The miner earns a block reward in the form of newly minted bitcoin, plus the transaction fees from the transactions included in the block.

So, how does this system prevent rewriting the past? Because each block's hash is part of the next block, meaning that changing a single transaction changes the block's hash and immediately breaks every block that comes after it.

> **Example** Imagine that you have two successive blocks, A and B. A's hash is `5` and B's hash is `6`. If you change a transaction in A, now A's hash has changed, and requires B's hash to change as well. B's hash takes into account A's hash given that B comes after A and A's hash is in B. So, B's hash will no longer be `6` if a transaction is changed in A.

In order to make the chain valid again, an attacker would have to redo the Proof of Work (the process of finding a nonce below a certain target value etc.) for not only that block, but for every subsequent block as well. Meanwhile, honest miners are mining and extending the "real" chain with new blocks. Additionally, Bitcoin follows the chain with the most cumulative Proof of Work, making it strongly inhibitive for attackers.

Therefore, a successful attack would require an attacker to have 51% of the mining power in order to eventually catch up with and become the 'real' chain. Mining power can also be referred to as *hash power*, as miners effectively just hash information countless times every second of every day.

The Transparency Tradeoff Importantly, for this system to function without a central authority, everyone must be able to verify everything. Every node checks every transaction against the full history of the chain, every UTXO is tracked, and every signature is validated.

This comes at the cost of privacy, as every transaction and address balance is public. The entire flow of funds, from the 2009 genesis block to the most recently mined block, is visible to anyone who downloads the blockchain.

So, Bitcoin solved the problem of trustless digital money, but it didn't solve the problem of trustless private digital money. That's where Zcash comes in.

3.2 Bitcoin, But Private

Zcash is effectively like Bitcoin, but with the addition of encryption. In fact, many refer to it as *encrypted Bitcoin*, even though it's a completely different cryptocurrency.

The economics of Zcash are nearly identical to Bitcoin's, so if you understand Bitcoin's monetary policy, you understand Zcash's as well. Zcash has a hard cap of 21 million ZEC, just like Bitcoin has a 21 million BTC hard cap. New coins enter circulation through mining rewards, which halve approximately every four years, as with Bitcoin.

The consensus mechanism is also Proof of Work, though Zcash uses Equihash rather than Bitcoin's SHA256-based system for mining. Something interesting about Equihash is that it was built with the explicit aim of resisting the specialized ASICs that dominate Bitcoin mining, therefore keeping mining accessible to people with consumer GPUs. The choice reflects Zcash's early emphasis on decentralization, though it no longer works as Equihash ASICs now exist.

> **Note** ASIC stands for *Application-Specific Integrated Circuit*, you can think of them as computers specifically designed to mine cryptocurrencies. There exist ASICs specialized in SHA256 mining, Equihash mining, etc.
>
> ASICs hash information (blocks of transactions) all day long in hopes of finding a hash below the network's target value.

Under the hood, Zcash uses the same UTXO transaction model as Bitcoin.

However, Zcash differs from Bitcoin in what you can do with the UTXOs. Bitcoin has one pool of funds: the public chain, whereas Zcash has several, split into the transparent pool and the shielded pools, but both pools use ZEC as currency, and you can move funds between them. The transparent pool works exactly like Bitcoin: addresses start with t, transactions are fully visible, and anyone can trace the flow of funds.

The shielded pools are completely different and are unique to Zcash. There are three pools , *Sprout*, *Sapling*, and *Orchard*, with Orchard being the newest and most advanced. Sprout and Sapling are now practically unused, since they date back to previous network upgrades and rely on *trusted setups*, which Orchard doesn't; we will cover this further later on in the article. Shielded addresses start with z, and transactions reveal nothing about the sender, the recipient, or the amount.

> **Note** Henceforth, we will refer to Zcash pools as the transparent pool and the shielded pool, as though there are several shielded pools, in practice they are consid-

ered as a unified whole and Orchard one primarily used today.

The transparent pool exists for compatibility and optionality. Some users want auditability, some applications even require it, and exchanges often use transparent addresses for regulatory compliance. In this case, transparency is a feature, not a bug, and Zcash's reliance on encryption for privacy in the shielded pool is unaffected by the usage of the transparent pools.

We should think of the transparent pool and the shielded pool as two entirely independent systems that do not affect each other. People often mistakenly criticize Zcash's transparency feature as somehow decreasing its privacy, but that is false. The Zcash anonymity set is mathematically independent from how much ZEC sits in transparent addresses. So, even if 99% of ZEC were transparent, the privacy of the shielded 1% would only be determined by the shielded pool itself.

3.3 The Fundamental Problem

In Bitcoin, validating a transaction is straightforward. You check that the inputs exist and haven't been spent before, that the signatures are valid, and that the outputs don't exceed the inputs. Every piece of information needed to verify these conditions is right there on the blockchain, visible to everyone.

Such transparency is what makes Bitcoin trustless. You don't need to trust anyone because you can verify everything yourself. If you wanted to, you could even run a node for maximal trustlessness. However, this is also what makes Bitcoin a surveillance tool, as the very data that enables verification is the same data that enables tracking.

Zcash wants both: trustless verification and privacy, but these seem to contradict each other. How can the network verify that a transaction is valid if it can't see the transaction?

Think about what validation actually requires:

1. The inputs exist, as you can't spend coins that don't exist.
2. The inputs haven't been spent before, so that there's no double-spending.
3. The authorization to spend, since you control the private key.
4. The math works out, and outputs don't exceed inputs.

In Bitcoin, nodes and miners check these four criteria by looking at the data. In Zcash, the sender, recipient, and amount are encrypted, and the data isn't visible. How then can anyone check these criteria?

The answer is that Zcash doesn't ask nodes and miners to check the data. Instead, the sender provides a zk-SNARK, a cryptographic proof, that demonstrates that the transaction is valid without revealing any of the underlying information. Miners and nodes don't learn what the inputs are, who the recipient is, or how much is being transferred, they only learn one thing: the proof is valid, and therefore the transaction is valid.

It sounds insane, we can verify a financial transaction is valid, without seeing it!

The following sections explain why this is possible, including how Zcash represents value and tracks what is spent, as well as how zero-knowledge proofs tie everything together.

3.4 Shielded Notes

As mentioned above, Bitcoin uses UTXOs. Zcash's shielded pool uses something conceptually similar called notes; you can think of notes as encrypted UTXOs.

So what is a note? A note is an encrypted object representing a specific amount of ZEC. It's a discrete chunk of value, just like UTXOs, but unlike UTXOs, its contents are hidden. When you receive shielded ZEC, a note is created. When you spend the shielded Zec, that note is consumed and new notes are created for the recipient and your change if applicable, exactly as with UTXOs.

Example This is what an Orchard note looks like after decryption:

```
{
  "addr":
    "u1pg2aaph7jp8rpf6...sz7nt28qjmxgmwxa",
  "v": 150000000,
  "rho":
    "0x9f8e7d6c5b4a...f8e7d6c5b4a39281706f5e4d3c2b1a0",
  "psi":
    "0x1a2b3c4d5e6f70...c4d5e6f708192a3b4c5d6e7f809",
  "rcm":
    "0x7a3b4c5d6e7b...d8e9f0a1b3d4e5f6a7b8c9d0e1f2a3b"
}
```

In this example, the value field v field shows 1.5 ZEC (150,000,000 zatoshis). The other fields, `rho`, `psi` and `rcm` will be covered later, for now, just understand that they are what makes the cryptography backing Zcash notes possible.

Notes are never modified, there is no updating of a balance. Rather, they're created, they exist, and they're destroyed when spent. If you have 10 ZEC and spend 3 ZEC, the original 10 ZEC note is consumed entirely, and two new notes are created: 3 ZEC given to the recipient and 7 ZEC returned to you, just like UTXOs.

The critical difference between Zcash's notes and Bitcoin's UTXOs is their visibility. A Bitcoin UTXO is public: everyone can see its value, when it gets spent, etc. A Zcash note is encrypted: only the owner, and anyone they share their viewing key with can see its contents. The blockchain stores a cryptographic commitment to the note, it does not store the note itself.

> **Example** The blockchain never sees the decrypted note. In Orchard, each 'action' bundles together a spend and an output. Here's what's actually recorded:


```
{
  "cv":
    "0x9a8b7c6d5...8d7e6f5a4b3c2d1e0f9a8b",
  "nullifier":
    "0x2c3d4e5f6a7b...d2e3f48e9f0a1b2c3d",
  "rk":
    "0x5e6f7a8b...5a6b7c8d9e0f1a2b3c4d5e6f",
  "cmx":
    "0x1a2b3c4d5e6f7...d3e4f5a6b7c8d9e0f1a2b",
  "ephemeralKey":
    "0x4d5e6f7a8b9...4f5a6b7c8d9e0f1a2b3c4d5e",
  "encCiphertext":
    "0x8f7e6d5c4b3...a29180f7e6d5c",
  "outCiphertext":
    "0x3c4d5e6f7a8...b9c0d1e2f3a4b5c"
}
```


> As you can see, it's all encrypted, we will go over the specifics of each field later.

You may be thinking, if notes are hidden, how does the network know they exist? Or how does it know when they've been spent? Here's where commitments and nullifiers come in.

3.5 Commitments and Nullifiers

Zcash's shielded pool faces two problems that Bitcoin solves trivially through transparency:

1. **Proving notes exist:** When someone sends you shielded ZEC, how does the network know the note is real?
2. **Preventing double-spending:** When you spend a note, how does the network know you haven't spent it before?

The solution for Zcash is a combination of two cryptographic mechanisms: commitments and nullifiers.

Commitments A commitment is a value computed by hashing the note's fields together. Here's what it looks like In Orchard:

```
cmx = Hash(addr, v, rho, psi, rcm) =
    0x1a2b3c4d...9ca6b7c8d9e0f1a2b
```

'Hash' denotes the hashing function used. We take the fields of the shielded note, feed them to the hash function, and it returns a hash (in this case `0x1a2b3c4d...9ca6b7c8d9e0f1a2b`).

There are two properties that make this useful:

1. **One-way:** given the returned hash, `0x1a2b3c4d...9ca6b7c8d9e0f1a2b`, you cannot recover the fields `addr`, `v`, `rho`, `psi`, or `rcm`, and the content of the note is hidden.
2. **Collision-resistant:** you cannot find two different notes that produce the same commitment, each note maps to exactly one commitment.

Each time that a note is created, its commitment is added to the commitment tree — a Merkle tree — containing every note commitment ever created on the network.

> **Info** A Merkle tree is a data structure that lets you prove that an item is in a large set without revealing the item or downloading the entire set.
>
> Here's how it works. Start with a list of values (in our case, note commitments): `cm0 cm1 cm2 cm3`
>
> Pair them up and hash each pair together:
>
> - `H0 = Hash(cm0, cm1)`
> - `H1 = Hash(cm2, cm3)`
>
> Now you have two hashes. Pair and hash again:
>
> `root = Hash(H0, H1)`
>
> So far, we have taken pairs of items from the original set and combined each pair using a hash function. We then group the resulting hashes into pairs and hash them again, repeating this process layer by layer until we reach a single final hash. This final value is called the root hash, or Merkle root.
>
> This root hash effectively summarizes the entire set:

```
             root
            /    \
           /      \
         H0        H1
        / \        / \
       /   \      /   \
     cm0   cm1  cm2   cm3
```

The key property of Merkle trees is that if you change any leaf (commitment), meaning the values `cm0`, `cm1`, etc., every single hash above it changes too, all the way back to the root. The root acts as the fingerprint of the entire tree, if you have the same root, then you must have the same tree.

Additionally, Merkle proofs provide an efficient way to check for an item in the tree without having to check the whole tree.

For example, to prove that cm1 is in the tree doesn't require revealing all of the commitments. To do so, just provide a Merkle path, that is, the sibling hashes along the way to the root. For `cm1`, the Merkle path is `[cm0, H1]`.

Here's how a verifier could check that: 1. Take the first element in `[cm0, H1]`, meaning `cm0`, and hash it with `cm1`, the item we want to check, this gives us `H0: Hash(cm0, cm1) = H0` 2. Hash the output of the first step (`H0`) with the following item in `[cm0, H1]`, meaning `H1`. This gives us the `root` hash: `Hash(H0, H1) = root`.

If the result matches the known root, then we can conclude that `cm1` is in the tree, importantly, the verifier never sees `cm2` or `cm3`, it's not necessary for the verification.

The commitment tree contains every shielded note commitment ever created, equalling millions of leaves (commitments). So, when you spend a note, you prove (inside the zk-SNARK) that you know a commitment and the valid Merkle path to the current root, without revealing which commitment is yours.

The commitment tree is stored by nodes, as part of the chain state they maintain. Each block introduces new note commitments which nodes append to their local copy of the tree, updating the root accordingly.. The current root, known as the anchor, is what transactions reference when proving membership.

Nullifiers Commitments may solve the existence problem, but they also create a new one: how do you prevent spending the same note twice?

In Bitcoin, this is trivial, because when you spend a UTXO, you directly reference its transaction identification and output index, such that everyone can see the UTXO has been spent. If you try to spend it again, nodes will reject the transactions because the UTXO has been marked as consumed.

The same is not possible for Zcash. If spending a note required pointing to its commitment, it would reveal which commitment you're spending and link that note to all future transactions, thus breaching privacy.

In Zcash, the solution to prevent spending the same note twice is *nullifiers*. Nullifiers are values derived from a note, and can only be computed by the note's owner.

> **Example** Let's say that the commitment tree has 1 million notes, and one of these notes is yours, specifically 'commitment `0x1a2b...`'
>
> If spending the note required you to say "I'm spending `0x1a2b...`" then:
>
> Everyone knows that `0x1a2b...` is yours, and it's no longer just one of a million anonymous commitments. It's tagged as belonging to whoever made this transaction, and though they don't know what's in that commitment, it's still problematic that they know it's yours.
>
> Senders can now track you, as whoever created that note by sending you the ZEC knows the commitment they created. So, when you spend and point to it, they are able to observe that the payment has been spent, and learn when you moved your funds.
>
> Over time, the spending may become linkable. An observer might be able to correlate transactions based on spending patterns, timing, and destination, such that your commitments get clustered together as "probably the same person."
>
> Nullifiers resolve these issues. If you publish the nullifier `0x2c3d...`, which corresponds to the commitment `0x1a2b...`, it's impossible to compute the mapping of commitments to nullifiers without knowing your private key. The commitment remains anonymous in the Merkle tree, your spends cannot

be linked, and the sender can't tell if their payment was spent.

Here's an example of a nullifier In Orchard:

```
nullifier = Hash(nk, rho, psi) =
    0x2c3d4e5f6a7b...d2e3f48e9f0a1b2c3d
```

nk is the nullifier deriving key, a secret key only you possess. `rho` and `psi` are values from the note itself, as seen previously. No one else can compute this nullifier because no one else has your `nk`. `Hash`, as in previous examples, is the hashing function being used (we will cover this later).

Anytime that you spend a note, you also publish its nullifier. The network maintains a nullifier set, that is, a collection of every nullifier ever published. So, if a nullifier is already in the set, the transaction gets rejected, thus preventing double-spending.

Example Here's how the nullifier set grows over time:

- Block 1000000: `nullifier set = { }`
- Block 1000001: `nullifier set = { 0x2c3d...3d }`
- Block 1000002: `nullifier set = { 0x2c3d...3d, 0x8f7a...2b }`
- Block 1000003: `nullifier set = { 0x2c3d...3d, 0x8f7a...2b, 0x1e4c...9a }`

Each spend adds exactly one nullifier. The set cannot shrink, it only ever grows.

At the risk of being repetitive, let us cover once more why unlinkability is the critical property. The nullifier reveals nothing about which commitment it corresponds to. An observer sees a nullifier appear and knows that some note was spent, but can't tell which one. The commitment tree could contain millions of notes and the nullifier could correspond to any of them.

Putting it all together Given that commitments are never deleted, as the commitment tree is append-only and grows indefinitely, commitments remain in the tree even after a note is spent.

This is precisely what makes Zcash's anonymity set so strong. Spending requires proving "I know one of the N million commitments in this tree" without revealing which one. The spent note's commitment is mixed among the others, so that even if an observer sees a nullifier appear they could not narrow down which of the millions of commitments it corresponds to.

Your privacy set includes every shielded note ever created on the network.

To summarize, every shielded transaction involves:

1. Creating notes, which adds new note commitments to the commitment tree.
2. Spending notes, which publishes and adds a nullifier to the nullifier set.

In order to construct a transaction, you must provide a zk-SNARK that proves:

- You know a note with a commitment in the tree, via a valid Merkle path.
- You know the secret key needed to compute that note's nullifier.
- The nullifier you're publishing corresponds to that note.
- The amounts balance of the entire transaction; inputs equal outputs plus fee.

The network verifies the proof, checks whether or not the nullifier is in the set, and accepts the transaction. Importantly, it never learns which commitment was spent, who sent funds to whom, or how much was transferred.

3.6 Keys and Addresses

Bitcoin has a simple key model: one private key, one public key, and one or more addresses. Zcash's shielded system is more complex, as different operations require different levels of access. Zcash leverages a hierarchy of keys to address this complexity.

The Spending Key The spending key (`sk`) is your master secret, it's a very long and random number of 256 bits. Whoever has this can spend your funds, as everything else is derived from the spending key.

The Full Viewing Key The full viewing key (`fvk`), derived from the spending key, lets you see everything about your wallet's activity: incoming payments, outgoing payments, amounts, and memo fields, but it cannot handle spending.

The full viewing key is useful for cases where you want to grant someone audit access without giving them control. Through the viewing key an accountant could verify your transaction history, a business could let compliance review its books, or a tax authority could confirm reported income; all without risking that the auditor walks away with the funds.

Incoming and Outgoing Viewing Keys The full viewing key can also be split into its constituent elements:

Incoming viewing key (`ivk`), which lets you detect and decrypt notes sent to you, but not notes that you've sent to others. Outgoing viewing key (`ovk`), which lets you decrypt the outgoing ciphertexts, so that you can see what you've sent and to whom.

This granularity exists because users may want to share only limited information. For example, if you want to provide a service with your incoming viewing key so the service can notify you of received payments, without revealing any information about your spending patterns.

The Nullifier Deriving Key The nullifier deriving key (`nk`), also derived from the spending key, is used to compute nullifiers when spending. This is required in order to mark notes as spent, which is why viewing keys alone can't authorize transactions—they don't have access to `nk`.

Addresses At the bottom of the hierarchy are the addresses: what you give to people so they can pay you. In Orchard, addresses are derived from the full viewing key using a diversifier, which is just a small piece of random data.

The diversifier enables diversified addresses, meaning you can generate billions of unlinkable addresses from a single wallet. Though each address is completely different, they all funnel to the same set of keys. Additionally, you can give a unique address to every person or service you interact with.

> **Example** Say you receive payments from an employer, a client, and an exchange. You give each a different diversified address:
>
> - Employer pays to: `u1employer8jp8rpf6...qjmxgmwxa`
> - Client pays to: `u1clientaph7jp8rpf...sz7nt28qj`
> - Exchange pays to: `u1exchng2aaph7jp8...gmwxasz7n`
>
> The three addresses belong to you and your wallet receives each sender's incoming payments, but the employer, client, and exchange cannot deduce that they're paying the same user by comparing their addresses.

The Key Hierarchy Here's the hierarchy:

```
spending key (sk)
|
+---> full viewing key (fvk)
|         |
|         +---> incoming viewing key (ivk)
|         |
|         +---> outgoing viewing key (ovk)
|         |
|         +---> addresses (via
diversifiers)
|
+---> nullifier deriving key (nk)
```

As you move down the hierarchy, each level reveals less information. The spending key can do everything, the full viewing key sees everything, but can't spend, and the incoming viewing key only sees incoming funds. Lastly, addresses reveal nothing, they're just destinations.

Figure 4: Eli Ben-Sasson, co-founder of Zcash and now leading StarkWare.

4. Transaction Lifecycle

This chapter will cover exactly what happens when you send shielded ZEC, from the moment you hit 'send' to the moment the recipient sees their balance update. To exemplify this, we'll follow every stage of a single transaction, examining what your wallet computes, what the network sees, and what ends up on the blockchain.

4.1 The Setup

Alice wants to send 5 ZEC to Bob. She opens her wallet, enters Bob's shielded address, specifies the amount, and confirms the send. What happens next involves each of the mechanisms we've covered thus far: notes, commitments, nullifiers, keys, Merkle proofs, and zk-SNARKs.

Alice's wallet holds two unspent notes:

- **Note A:** 3 ZEC
- **Note B:** 4 ZEC

She'll spend both (7 ZEC total) to send Bob 5 ZEC, pay a 0.001 ZEC fee, and receive 1.999 ZEC in change.

4.2 Note Selection and Retrieval

Remember, Alice's wallet doesn't actually store ZEC, it stores the information needed to spend notes: the decrypted note data and the keys that control them. When Alice synced her wallet, it scanned the blockchain, attempted to decrypt every shielded output using her incoming viewing key, and stored the ones that succeeded.

Here's an example of note A:

```
{
  "addr": "u1alice...",
  "v": 300000000,          // 3 ZEC in
  zatoshis
  "rho": "0x7a8b9c...",
  "psi": "0x1d2e3f...",
  "rcm": "0x4a5b6c...",
  "position": 847291,      // Position in
  commitment tree
  "cmx": "0x9f8e7d..."     // The commitment
}
```

The position field is crucial because it tells the wallet where in the commitment tree this note is situated, information necessary to construct the Merkle proof.

4.3 Fetching Merkle Paths

In order to spend a note, Alice must prove that its commitment exists in the tree, without revealing which commitment it is. This requires proving a Merkle path from the commitment to the root.

Alice's wallet queries a full node for the Merkle path at each note's position. For Note A, at position 847,291 in a tree with depth 32, the path consists of 32 sibling hashes:

```
merkle_path_A = [
  "0x1a2b3c...",  // Sibling at level 0
  "0x4d5e6f...",  // Sibling at level 1
  ...            // 30 more siblings
  "0x7g8h9i..."   // Sibling at level 31
]
```

Anyone with access to this path can verify that `cmx_A` is in the tree by hashing back to the root. but, inside the zk-SNARK, Alice can prove this without revealing `cmx_A` or the path itself.

The wallet also records the anchor—the Merkle root at the time of path retrieval. The transaction will reference this anchor and nodes can use it to verify that it's a recent, valid root.

4.4 Computing Nullifiers

Alice has her notes and their Merkle paths, now she needs to mark them as spent.

Recall from section 3.5 that nullifiers solve the fundamental problem of preventing double-spending without revealing the note being spent? With Bitcoin, you have to point to a UTXO directly and everyone can see it's now consumed, but with Zcash, pointing to a commitment would destroy privacy by linking you to that specific note.

Alice computes a nullifier for each note she's spending, the nullifier is derived from the note's data and her secret nullifier deriving key (`nk`):

```
nullifier_A = Hash(nk, rho_A, psi_A) =
    0x2c3d4e5f...
nullifier_B = Hash(nk, rho_B, psi_B) =
    0x8f7a9b2c...
```

The `rho` and `psi` values are unique to each note, meaning they were set when the note was created. The `nk` is derived from Alice's spending key, but only she possesses it.

The construction has two critical properties:

1. **It's deterministic:** Each note produces exactly one nullifier. If Alice tried to spend Note A twice, she'd have to publish `0x2c3d4e5f...` twice. The network maintains a nullifier set of every nullifier

ever published, so the second attempt would be rejected because that nullifier already exists.

2. **It's unlinkable:** No one else can compute the nullifier for Alice's notes because no one else has her nk, and crucially, no one can work backwards from a nullifier to determine its corresponding commitment. So, when `0x2c3d4e5f...` appears on the blockchain, observers will see that some note was spent, but won't be able to tell which of the millions of commitments in the tree it came from.

The nullifiers will be included in Alice's transaction and published on-chain, but are the only public trace of her spending. Just two opaque 32-byte values that reveal nothing about the notes themselves, their amounts, or who controlled them.

> **Note** The nullifier set only grows. Unlike the commitment tree (which is append-only but tracks all notes ever created), the nullifier set tracks spent notes. A note's commitment stays in the tree forever, even after it's spent. The nullifier's presence in the nullifier set is what marks it as consumed.

4.5 Creating Output Notes

Alice is spending 7 ZEC (3 ZEC + 4 ZEC) and needs to create two new notes: 5 ZEC for Bob and 1.999 ZEC for her change; there's a 0.001 ZEC transaction fee.

Each note requires novel randomness, so Alice's wallet generates the cryptographic components that make each note unique and spendable only by its intended recipient.

Generating Note Components For Bob's 5 ZEC note:

```
{
    "addr": "u1bob...",           // Bob's
    shielded address
    "v": 500000000,               // 5 ZEC in
    zatoshis
    "rho": "0x3e4f5a6b...",       // Derived
    deterministically
    "psi": "0x7c8d9e0f...",       // Random
    "rcm": "0x1a2b3c4d..."        // Random
    (commitment randomness)
}
```

For Alice's 1.999 ZEC change note:

```
{
    "addr": "u1alice...",         // Alice's
    own address
    "v": 199900000,               // 1.999
    ZEC in zatoshis
    "rho": "0x5f6a7b8c...",
    "psi": "0x9d0e1f2a...",
    "rcm": "0x4e5f6a7b..."
}
```

The `rho` value in Orchard is derived deterministically from the transaction, which prevents against certain types of cryptographic attacks. The `psi` and `rcm` values are freshly sampled random numbers. Together, these values ensure that even if Alice sends Bob 5 ZEC a thousand times, the note's commitment would be different every time.

Computing Commitments Once the note components are ready, Alice computes the commitment for each output:

```
cmx_bob = Hash(addr_bob, 500000000, rho_bob,
    psi_bob, rcm_bob)
                   = 0x8a9b0c1d...
```

```
cmx_alice = Hash(addr_alice, 199900000,
    rho_alice, psi_alice, rcm_alice)
                   = 0x2d3e4f5a...
```

These commitments are what will be published on-chain and added to the commitment tree. They reveal nothing about the notes themselves, they are opaque 32-byte hashes, but anyone who knows the underlying values (the recipient, specifically), can verify that a commitment corresponds to a specific note.

Encrypting the Notes The commitments go on-chain, but Bob needs the actual note data in order to later spend his 5 ZEC. He needs to know the value, `rho`, `psi`, and `rcm`, as without these, the commitment is useless as he can't construct a valid nullifier or prove ownership.

Alice encrypts each note so that only the intended recipient can read it:

For Bob: Alice uses Bob's address (which contains his public key material) to encrypt the note. The result is the `encCiphertext` ciphertext: a blob of encrypted data that can only be decrypted using Bob's incoming viewing key. When Bob's wallet scans the blockchain and successfully decrypts this ciphertext, he learns he received 5 ZEC and stores all the data needed to spend it.

For Alice's records: There's a second ciphertext called `outCiphertext`: this one is encrypted to Alice's outgoing viewing key, allowing her wallet to remember what she sent. Without this, Alice wouldn't have a record of where her funds went. It's encrypted, rather than being stored in plaintext, so that node operators and observers can't read it.

```
{
    "cmx": "0x8a9b0c1d...",
    "ephemeralKey": "0x6b7c8d9e...",
    "encCiphertext": "0x9f8e7d6c5b4a...[512
    bytes]...",
    "outCiphertext": "0x3c4d5e6f7a8b...[80
    bytes]..."
}
```

The `ephemeralKey` is a one-time public key generated for this specific encryption, and Bob can use it alongside his private key in order to decrypt `encCiphertext`. This is standard for public-key encryption, but the twist is that it's happening inside a system which never linked Bob's address to an identity, and where the ciphertext doesn't reveal anything to outside observers.

> **Note** The encryption is not part of what the zk-SNARK proves. The encryption is a separate layer that ensures only recipients can access their funds, whereas the proof verifies that notes are correctly formed and that the transaction amounts balance. If Alice encrypted incorrectly (or maliciously used the wrong key), the transaction would still be valid on-chain—but Bob would never be able to find or spend his note. In practice, wallets handle this correctly, and the recipient's inability to decrypt would be a wallet bug, not a protocol violation.

At this point, Alice has everything required for the outputs: two commitments to publish and encrypted payloads so that each recipient can claim their note. Now comes the hard part: proving it's valid without revealing any of it.

4.6 The Proof

Alice has assembled all of the pieces: the two notes to spend, their Merkle paths, the nullifiers that will mark them as consumed, and two fresh output notes with their commitments and encrypted payloads. Now, how to convince the network that everything is valid without revealing the details?

Here's where zk-SNARK comes in.

What the Proof Demonstrates The proof is a cryptographic object that demonstrates all of the following are true:

1. **The input notes exist.** Alice knows two of the commitments that are in the commitment tree. She proves this when outlining the valid Merkle paths from those commitments to the anchor (the tree root). The proof doesn't reveal which commitments Alice is referencing, just that they're in there somewhere among the millions.
2. **Alice controls the inputs.** Alice knows the spending keys for both notes, specifically, she knows the secret values needed to derive the nullifiers and authorize the spend. Without this, anyone could try to spend anyone else's notes.
3. **The nullifiers are correct.** The nullifiers that she's publishing actually correspond to the notes she's spending. Alice can't publish arbitrary nullifiers, they must be derived from real notes she controls using the proper formula.
4. **The transaction amounts balance.** The sum of the input values ($3 + 4 = 7$ ZEC) equals the sum of the output values ($5 + 1.999 = 6.999$ ZEC) plus the fee (0.001 ZEC). No ZEC is created or destroyed. This is the fundamental conservation law of the system.
5. **The output commitments are well-formed.** The commitments she's publishing for Bob's note and her change note are correctly computed from valid note data. She can't publish garbage commitments—they must follow the proper structure.

The network doesn't learn which notes were spent, who the recipient is, or the amount that moved from one party to another. It only learns that someone made a valid transaction: real inputs, real outputs, correct math, and proper authorization. That's enough to update the global state, meaning adding commitments and recording nullifiers, without knowing anything about the transaction itself.

What the Proof Actually Is After all of this complexity, the proof itself is almost anticlimactic: roughly one to two kilobytes of data - that's it! It's just a small blob of bytes that encodes a mathematical argument.

Verification is fast, just a few milliseconds on modest hardware. A node receives the proof, runs the verification algorithm, and returns a binary answer:

valid or invalid. No judgment calls, no heuristics, no probabilistic guesses; the math either checks out or it doesn't.

This asymmetry is zk-SNARKs' magic. Creating the proof is computationally expensive, Alice's wallet does real work, crunching through elliptic curve operations and polynomial math. However, verifying the proof is cheap. The asymmetry makes the system practical: every node on the network can verify every shielded transaction without re-doing the heavy computation.

The Circuit How does Alice actually produce this proof? By running her transaction data through something called a circuit—a formal specification of exactly what conditions must hold for a valid Orchard spend.

Think of the circuit as a massive checklist encoded in mathematical constraints. The step to prove "the Merkle path must be valid" becomes a series of hash computations that must produce the right output, the step "the nullifier must be correctly derived" becomes constraints on how certain values relate to each other, finally "the amounts must balance" becomes an equation that must hold.

Alice's wallet takes her private inputs (notes, keys, paths, randomness) and grinds through this circuit to find values that satisfy every constraint. The zk-SNARK machinery then compresses this entire satisfying assignment into a tiny proof that anyone can check.

We'll dissect exactly how this works—through polynomial commitments, elliptic curves, and the Halo 2 proving system—in section 7. For now, what's important to know is that the proof is the output of a deterministic process: valid inputs produce a valid proof, and invalid inputs don't produce anything useful.

> **Note** The circuit is fixed at the protocol level, and every Orchard transaction uses the same circuit, as defined in the Zcash specification. Alice can't modify the rules, she can only prove that she followed them. This is what makes the system trustless: nodes don't need to trust Alice, they just need to verify that her proof passes the universal circuit agreed to.

Alice's wallet has now produced a proof: a ~1.5 KB object asserting that a valid transaction exists, without saying what it is. Now it's time to package everything up and send it to the network.

4.7 Assembling the Transaction

Alice has her nullifiers, her output notes, her encrypted payloads, and her proof - now she needs to package everything into a transaction that the network can process.

The Action Structure Orchard uses a structure called an action. Each action bundles exactly one spend and one output together, this is a deliberate design choice. Earlier Zcash protocols (Sprout and Sapling), separated spends and outputs, but this leaked information about transaction structure. If you saw a transaction with three spends and one output, you would be learning something. Orchard eliminates this problem by forcing a 1:1 pairing.

Alice is spending two notes and creating two outputs, so her transaction contains two actions:

- **Action 0:** Spends Note A (3 ZEC), creates Bob's note (5 ZEC)
- **Action 1:** Spends Note B (4 ZEC), creates Alice's change note (1.999 ZEC)

The pairing within each action is arbitrary. Action 0 doesn't mean Note A "became" Bob's 5 ZEC. The values don't match, and that's fine. What matters is the global constraint: total inputs equal total outputs plus fee. The action structure just ensures tat observers can't infer transaction shape.

> **Note** What if Alice wanted to spend two notes, but only create one output? In order to do this, she would still need two actions, so she would have to create a dummy output in the second action. A dummy is a zero-value note that exists only to balance the structure. The same applies in reverse: if she had one input but needed two outputs, she would include a dummy spend. Observers can't distinguish real actions from dummies.

What Goes Onchain Here's what Alice's transaction actually contains:

```
{
    "anchor": "0x7f8e9d0c...",

    "actions": [
        {
            "cv": "0x9a8b7c6d...",
            "nullifier": "0x2c3d4e5f...",
            "rk": "0x5e6f7a8b...",
            "cmx": "0x8a9b0c1d...",
            "ephemeralKey": "0x6b7c8d9e...",
```

```
            "encCiphertext":
    "0x9f8e7d6c...[580 bytes]",
            "outCiphertext":
    "0x3c4d5e6f...[80 bytes]"
        },
        {
            "cv": "0x1b2c3d4e...",
            "nullifier": "0x8f7a9b2c...",
            "rk": "0x4d5e6f7a...",
            "cmx": "0x2d3e4f5a...",
            "ephemeralKey": "0x8c9d0e1f...",
            "encCiphertext":
    "0x7e8f9a0b...[580 bytes]",
            "outCiphertext":
    "0x5a6b7c8d...[80 bytes]"
        }
    ],

    "proof": "0x1a2b3c4d...[~1.5 KB]",
    "bindingSig": "0x4e5f6a7b...[64 bytes]"
}
```

Let's break this down:

anchor: The Merkle root that Alice's proof references. This commits her transaction to a specific state of the commitment tree. Nodes will verify this is a recent, valid root. If Alice tried to use an anchor from a year ago, the transaction would be rejected.

- **cv (value commitment):** A cryptographic commitment to the value being spent or created in each action. These don't reveal the actual amounts. Instead, they're constructed so that the sum of all cv values across the transaction encodes the net flow. If the transaction is balanced (inputs = outputs + fee), the math works out. If not, verification fails.
- **nullifier:** The nullifiers for Note A and Note B. These get added to the nullifier set, marking those notes as spent forever.
- **rk (randomized verification key):** This is used to verify the spend authorization signature. This proves Alice authorized this specific transaction without revealing her actual spending key.
- **cmx:** The commitments for Bob's note and Alice's change note. These get added to the commitment tree.
- **ephemeralKey + encCiphertext + outCiphertext:** The encrypted note data, as covered in section 4.5. These don't affect consensus, but without them, recipients couldn't claim their funds.
- **proof:** The zk-SNARK proving everything is valid. One proof covers the entire transaction (both actions).
- **bindingSig:** A signature that ties all the pieces together. It proves that the cv values across all actions sum correctly (guaranteeing value conservation) and that the transaction hasn't been tampered with. This is the final check that the amounts actually balance.

The Fee You'll notice the fee isn't explicitly stated anywhere, that's because it's implicit. Alice's input total is 7 ZEC and her output total is 6.999 ZEC. The difference, 0.001 ZEC, is the transaction fee, which is claimed by miners.

The value commitments encode net flow, so when a miner verifies the binding signature, they're confirming that inputs minus outputs equals the claimed fee. If Alice tried to claim her outputs totaled 7 ZEC, leaving no fee, the binding signature would fail. If she tried to create extra ZEC out of thin air and claimed 8 ZEC of outputs from the 7 ZEC of inputs, the proof itself would be invalid.

The fee is public. Observers can see how much was paid to process the transaction, but that's the only visible value. The input amounts, output amounts, and transfer of value between parties remain hidden.

4.8 Broadcasting and Mempool

Alice's wallet has assembled the complete transaction, now it needs to reach the network.

Sending to the Network The sending process proceeds as follows. Alice's wallet connects to one or more Zcash nodes and broadcasts the transaction. The message propagates through the peer-to-peer network, hopping from node to node until it reaches the miners and the broader network. The sending process works exactly as in Bitcoin, the transaction is just data gossiped from nodes to peers.

From Alice's perspective, this takes one or two seconds. She sees "transaction broadcast" in her wallet and just waits for the confirmation.

Initial Validation When a node receives Alice's transaction, it doesn't blindly accept it. Before relaying it further or adding it to the mempool, the node runs a series of checks:

1. **Proof verification:** The node runs the zk-SNARK verifier on Alice's proof. This takes

a few milliseconds. If the proof is invalid, the transaction is rejected immediately. No further checks needed.

2. **Anchor check:** The node verifies that the anchor Alice used (the Merkle root her proof references) is valid. Specifically, it must be a recent root from the commitment tree. Zcash allows a window of recent anchors to accommodate network latency. If Alice's anchor is too old or doesn't match any known tree state, the transaction is rejected.
3. **Nullifier check:** The node checks both nullifiers against its local nullifier set. If either `0x2c3d4e5f...` or `0x8f7a9b2c...` already exists in the set, Alice is attempting to double-spend. The transaction is rejected.
4. **Structural validity:** The node confirms the transaction is well-formed: correct field lengths, valid encodings, binding signature verifies, and so on. Malformed transactions are dropped.

If all of the checks pass, the node considers the transaction valid. Then, it adds the transaction to its mempool (memory pool), a holding area for unconfirmed transactions, and relays it to other nodes.

Waiting in the Mempool The mempool is purgatory for transactions. Alice's transaction sits there alongside hundreds or thousands of others, all waiting for a miner to pick them up and include them in a block.

Miners select transactions from the mempool based on fees. Higher fee transactions generally get picked first. Alice paid 0.001 ZEC, which is typical for Zcash, and under normal network conditions, this is enough to get included in the next block or two.

During the waiting period, Alice's transaction is unconfirmed. The network has validated it, but it hasn't been written into the blockchain yet. Bob's wallet might detect the pending transaction - some wallets show incoming unconfirmed transactions - but he can't spend those funds until the transaction is mined.

> **Note** The mempool is not global, nor synchronized, each node maintains its own mempool. Due to network propagation delays, different nodes might have slightly different sets of pending transactions at any given moment. This doesn't matter for consensus, what does matter is which transactions make it into blocks.

The transaction is broadcast. Nodes have validated it. Now, Alice waits for a miner to do the final work.

4.9 Block Inclusion and Finality

A miner selects Alice's transaction from their mempool, bundles it with other transactions, and begins the work of mining a new block.

Mining the Block Zcash uses Proof of Work, just like Bitcoin. The miner constructs a block header containing the previous block's hash, a timestamp, a Merkle root of the included transactions, and a nonce. Then, they grind through nonces until finding one that produces a hash below the target difficulty.

This process is identical to what we covered in the Bitcoin primer (section 3.1), with one exception: Zcash uses the Equihash algorithm instead of SHA256. The security properties are the same - finding a valid block requires significant computational work and verifying that work is trivial.

When a miner finds a valid nonce, they broadcast the block and then other nodes verify it: valid proof of work, valid transactions, correct structure. If everything checks out, nodes append the block to their chain and Alice's transaction becomes part of the permanent record.

State Updates Once the block is accepted, the network's state changes:

- **The commitment tree grows:** Bob's note commitment `0x8a9b0c1d...` and Alice's change note commitment `0x2d3e4f5a...` are appended to the commitment tree. Now the tree now contains two more leaves than before and a new Merkle root is computed. This root becomes a valid anchor for future transactions.
- **The nullifier set expands:** Alice's two nullifiers (`0x2c3d4e5f...` and `0x8f7a9b2c...`) are added to the nullifier set. Those notes are now permanently marked as spent. Any future transaction attempting to use either nullifier will be rejected.
- **The block reward is issued:** The miner receives newly minted ZEC (the block subsidy) plus the sum of all transaction fees in the block, including Alice's 0.001 ZEC.

These state updates are deterministic. Every node that processes the block arrives at exactly the same new state. The commitment tree has the same new root everywhere. The nullifier set contains the same entries everywhere. This is what makes the network consistent without central coordination.

Confirmations Alice's transaction is now confirmed, but the confirmation doesn't mean finality.

Just like Bitcoin, Zcash can experience temporary chain forks. Two miners might find valid blocks at nearly the same time, causing the network to briefly disagree on which chain is correct. The longest chain, meaning the one with the most cumulative proof of work, is the one that wins. Transactions in the losing fork return to the mempool and may be re-mined later, or they may become invalid if they conflict with transactions in the winning chain.

The probability of a transaction being reversed drops exponentially with each additional confirmation. After one block, there's a small chance of reorganization. After six blocks, the chance is negligible for practical purposes. For large transfers, recipients often wait for multiple confirmations before considering the payment final.

Alice's wallet sees the confirmation count tick upward as new blocks are mined on top of the one containing her transaction. After a few minutes, she can be confident the transaction is permanent.

> **Note** Zcash has a block time of 75 seconds, compared to Bitcoin's 10 minutes, so confirmations accumulate much faster. Six confirmations take about seven and a half minutes for Zcash, versus an hour for Bitcoin. The tradeoff is that each individual Zcash block represents less total work, so more confirmations may be prudent for very high-value transfers.

The transaction is mined and the state is updated. Alice's old notes are gone forever, replaced by two new notes in the commitment tree. One of those notes belongs to Bob, and now he needs to find it.

4.10 Recipient Detection

Alice's transaction is on-chain. Bob's 5 ZEC exists as a commitment in the tree, but Bob doesn't know that yet. His wallet needs to find the corresponding note.

Scanning the Blockchain Bob's wallet periodically syncs with the network, downloading new blocks and scanning for incoming payments. The challenge is that Bob can't simply search for his address. Shielded outputs don't contain addresses in plaintext, every output resembles random encrypted data.

Bob's wallet tries to decrypt every shielded output it encounters, so for each `encCiphertext` of every action of every block, the wallet attempts decryption using Bob's incoming viewing key. Most of these attempts fail and produce unusable data, but that's expected since those outputs belong to someone else.

Finally, when Bob's wallet hits Alice's transaction and tries to decrypt the ciphertext in Action 0, the decryption succeeds and the valid note data emerges.

Recovering the Note When decryption works, Bob's wallet recovers the full note plaintext:

```
{
    "addr": "u1bob...",
    "v": 500000000,
    "rho": "0x3e4f5a6b...",
    "psi": "0x7c8d9e0f...",
    "rcm": "0x1a2b3c4d..."
}
```

Bob now has everything he needs:

- **The value:** 5 ZEC (500,000,000 zatoshis). His wallet updates his balance accordingly.
- **The note components:** The `rho`, `psi`, and `rcm` values that Alice generated. These are essential. Without them, Bob couldn't compute the commitment to verify that it matches what's on-chain, or derive the nullifier to spend the note later.
- **The position:** Bob's wallet also records where this commitment sits in the tree. When the block was processed, the commitment was appended at a specific leaf index. Bob needs this position to construct a Merkle path when he eventually spends.

Verifying the Note Bob's wallet doesn't blindly trust the decrypted data. It recomputes the commitment from the recovered values:

```
cmx_check = Hash(addr_bob, 500000000, rho,
    psi, rcm)
```

If `cmx_check` matches the `cmx` published onchain in Alice's transaction, the note is valid. If they don't match, something is incorrect (either corruption or malicious senders), and the wallet discards the note.

During normal operations, this check always passes. Alice's wallet constructed the note correctly, and the decryption recovered exactly what she encrypted.

A Spendable Note Bob now owns a spendable 5 ZEC note. His wallet stores the note data locally and keeps it ready for whenever he wants to use it. At that point, he'll follow the same process that Alice did in order to send it to him:

1. Select the note
2. Fetch its Merkle path
3. Compute its nullifier
4. Create output notes for his recipients
5. Generate a proof
6. Broadcast the transaction

The cycle repeats: Bob's spend will reveal a nullifier, marking his note as consumed, new commitments will be added to the tree, and then new recipients will scan, decrypt, and discover their funds.

> **Note** Scanning is the main performance bottleneck for shielded wallets, as a wallet that's been offline for months needs to trial-decrypt millions of outputs to catch up. It's for this reason that light clients and optimized sync protocols matter. Project Tachyon, mentioned in section 2, aims to dramatically improve the catch-up process with oblivious synchronization, letting wallets query servers for relevant data without revealing what information is being sought.

Alice sent 5 ZEC to Bob. The network verified the transaction without learning who sent what to whom, but Bob was still able to detect his payment without anyone else knowing he received it. The transaction is complete.

5. The Philosophy of Privacy

5.1 Privacy as a Precondition for Progress

Privacy does not mean secrecy, for secrecy aims to hide something shameful. Privacy is the right to choose what you reveal and to whom. Privacy is autonomy over your own information, it's the foundation of freedom itself.

This distinction matters because critics of privacy often conflate the two. The refrain of authoritarian systems proclaims that "if you have nothing to hide, you have nothing to fear," and assumes that privacy is only valuable to those with something to conceal. However, privacy is valuable to everyone, precisely because it creates the conditions for everything else we value: free thought, free speech, free markets, and progress.

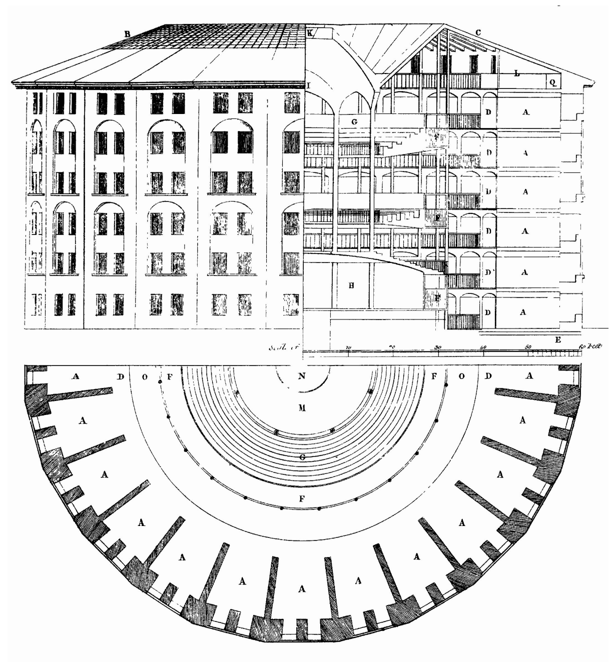

Figure 5: Jeremy Bentham's Panopticon, 1791. A prison designed so inmates never know if they're being watched. They learn to watch themselves.

The Conditions for Progress Karl Popper thought that progress was dependent on criticism. Bold ideas must be proposed, tested, and corrected, so that errors can be identified and discarded. This process requires freedom from punishment for proposing bold ideas before they're tested and which risk being wrong. The panopticon ensures that dissent is muted before it can be voiced. Innovation requires permissions and criticism gets punished - the mechanism of progress breaks down.

David Deutsch extended Popper's insights, positing that humans are unique in virtue of being universal explainers. Our capacity to create knowledge, to understand the cosmos, and even transform them, makes us special. Yet, knowledge creation requires experimentation, and experimentation requires the freedom to fail privately before succeeding publicly. Surveillance inhibits the freedom to experiment, for when every action is observed and recorded, it suffocates creative thinking.

These are not abstract concerns. They're the reality of anyone who has self-censored knowing their words were being surveilled. Anyone who chose not to donate to a controversial or novel cause knowing the transaction would be visible or traceable. Anyone who avoided researching a sensitive topic knowing the query would be logged. Surveillance changes behavior, that's one of its primary functions. Sometimes, changing behaviors amounts to constraining ideas, and thus, to constraining progress.

Money as the Final Monopoly Throughout history, freedom has depended on the tools we had to protect it.

The printing press encouraged free speech. Before Gutenberg, ideas often were chained to scribes and priests, locked behind institutional authority. The press broke the monopoly on information.

The internet broke the monopoly of geography. Ideas could now be shared across borders in an instant. Coordination became possible without physical proximity. Censorship became harder when information could route around obstacles.

Gunpowder shattered the monopoly of knights and kings over violence. A peasant with a musket could challenge a lord in armor. Power became more distributed.

Every time, a new tool smashed an old monopoly. Now, one monopoly remains: money.

Money is the most powerful coordination technology humans ever built. It's how we signal value, allocate resources, and cooperate at scale, but it remains significantly restricted. Money is the most surveilled and the most controlled technology. Every transaction can be monitored. Governments can freeze accounts with a keystroke. Banks can cancel you overnight. Increasingly, capital controls can prevent you from withdrawing your own cash.

Arguably, your money is not yours if someone else can see every transaction you make, decide whether or not to approve it, and later even decide to reverse that decision and inhibit access.

Privacy in Markets Free markets require privacy, this conclusion follows from understanding how markets work:

Markets aggregate information through prices, meaning that as participants make decisions based on private knowledge, prices emerge from the sum of those decisions. The mechanism functions only if participants are able to make decisions based on their private information without also revealing it prematurely. A trader who must broadcast every position before taking it will be front-run. A business that must publish every supplier relationship will be undercut. A donor who must announce every contribution will be pressured.

Any leak, even just of small pieces of information, changes the market because it introduces bias and distorts decisions. The more surveillance there is in a system, the more distortion it faces. Perfect markets require participants who can act freely on private information, which is made impossible under conditions of absolute surveillance.

Your net worth should not be a public API. Your transaction history should not be a queryable database. Your financial life should not be subject to the approval of observers. These are not edge cases or paranoid concerns, they are the baseline requirements for markets to function and for individuals to be free.

5.2 The Transparency Trap

Crypto was supposed to free us from financial surveillance, but It did the opposite.

The cypherpunks who built this movement understood the stakes at hand. They understood that privacy in the digital age would not be granted by governments or corporations, but would have to be built, deployed, and defended with cryptographic tools. Bitcoin emerged from this tradition, and it did succeed in being the first crack in the dam, proof that money could exist outside government control.

However, Bitcoin has a major flaw, it's transparent by default. Every transaction, every address, and every balance is visible to anyone interested in looking for it. The blockchain is a permanent public ledger of all of the economic activity that has ever come across it. Satoshi acknowledged this limitation in his original whitepaper, suggesting that users could preserve some privacy by generating new addresses for each transaction. That was a weak mitigation then, and it's developed into an absurd one now.

Pseudonymity means that your identity is not directly tied to your address. Nevertheless, your identity can leak through observation of your behavior, such as the times at which you transact, the amounts that you move, the addresses you interact with, in sum, the patterns you repeat. With each data point the set of possible identities for an address narrows, until finally, with enough constraints, the set collapses to one.

In the age of AI, pseudonymity is privacy on borrowed time, it's just an illusion waiting to be dissolved by compute.

Commerce Requires Opacity The transparency problem is not limited to individuals, as commerce also breaks down without privacy.

Consider what happens when you make a single payment to a business on a transparent chain. Now, you can now see their address, and from that address potentially derive their total revenue, their customer addresses, their supplier relationships, their payroll, even their cash flow and runway.

There is a reason that HR departments treat compensation structures as closely guarded secrets, that businesses do not publish their supplier contracts, and that financial statements are released quarterly, in controlled formats, rather than streamed in real-time to the public. Competitive markets require informational asymmetry. Businesses must be able to act on private knowledge without broadcasting it to competitors.

The same logic applies to individuals. If your spending patterns reveal your health conditions, your political affiliations, your religious practices, and your personal relationships, then every transaction becomes a data point in generating a picture of who you are, what you value, and how you can be influenced or coerced.

The web needed HTTPS before commerce could function online. Transmitting credit card numbers in plaintext was obviously unacceptable due to security reasons. The payment layer of the internet needs the same evolution, just as plaintext transactions were a prototype, production requires encryption.

5.3 Privacy Must Be Absolute

Half-measures do not work because privacy is binary - you either have it or you do not.

This may sound extreme, but it follows from how information works. A secret is only a secret until it leaks, as once it's leaked, it cannot be unleaked. In a world of permanent storage, pattern recognition, and AI-powered analysis, any partial leak grows to become a full leak. The question is not if the remaining bits of information will be extracted, but when.

The Single-Bit Problem Imagine a privacy system that hides 99% of your transaction data but leaks the remaining 1%. That 1% might seem acceptable, but information compounds. One leaked bit constrains possibilities, and two bits constrain it further. Each additional leak narrows the possibilities of who you could be, what you could be doing, and why.

Adversaries are patient. They're going to collect partial pieces of information over time, correlate across data sources, and apply statistical techniques to extract the signal from the noise. Though a timing correlation here, an amount pattern there, a network graph connection elsewhere are not individually sufficient to identify you, they can once they converge.

Remember, this is not a hypothesis, it's the methodology of chain analysis, metadata analysis, and every modern surveillance system. The assumption that small leaks will remain small is incorrect; small leaks accumulate to compose complete pictures.

Any privacy system that leaks must answer the following question: What happens when an adversary with unlimited time and compute optimizes against those leaks? If the answer is "they eventually win," then the system does not provide privacy, it just provides delayed exposure.

Obfuscation vs Encryption There are two approaches to hiding information, either you obfuscate it, making it harder to find among noise, or you encrypt it, making it mathematically inaccessible without the key.

Obfuscation is hiding a needle in a haystack. It works until someone builds a better magnet. The needle is still there, still findable with sufficient effort. The security is economic, not mathematical. You are betting that finding the needle costs more than it is worth. But costs decline over time. Compute gets cheaper. Algorithms get smarter. Adversaries get more motivated. What is hidden today may be trivially exposed tomorrow.

Encryption is destroying the needle and keeping only a locked description of it. Without the key, the description is indistinguishable from random noise. There is no magnet that helps. There is no amount of computation that extracts meaning from randomness. Security is mathematical, not economic. It does not degrade over time. An encrypted message from 2016 is exactly as secure today as it was then, assuming the cryptography was sound.

This distinction matters enormously for financial privacy. Obfuscation-based approaches mix your transaction with others, to hide it among decoys or add noise to the data. Although these techniques raise the cost of analysis, they do not make analysis impossible. As analysis techniques improve, the protection weakens. Privacy that was adequate five years ago may be broken today, and privacy that seems adequate today may be broken by the tools of

2030.

Encryption-based approaches hide the transaction itself, there is no transaction to analyze, only a proof that a valid transaction occurred. The data is not simply obscured, it is absent, and therefore immune to future developments in analysis techniques; you cannot find patterns in data that do not exist.

Why This Determines Architecture This is why Zcash encrypts transactions rather than obfuscating them. The sender, recipient, and amount are not hidden among decoys or mixed with noise. Instead, they are encrypted. The blockchain stores commitments and proofs, not obscured data, so what the network sees is mathematically indistinguishable from random bytes.

The concise argument is that if you accept that privacy must be absolute, that partial leaks compound into total exposure, and that adversary capabilities only grow over time, then encryption is the only viable architecture as the permanent solution, and obfuscation is just a temporary measure.

The choice is not between more privacy and less privacy. It is between privacy that will hold and privacy that will eventually fail. There is no middle ground.

5.4 The Macro Case

So far, the privacy arguments have been philosophical. That privacy enables progress, transparency equates to surveillance, and partial privacy fails, remain true in any era. However, we do not live in just any era, in the current era, the macro environment makes privacy not just valuable but urgent.

History Does Not End The stability of modern western societies may have led people to misjudge the permanence of stability. Throughout history and across the world, stability is the exception, not the rule. Regimes collapse. Currencies fail. Debt cycles reset. Capital controls appear overnight. These are not rare events or distant history, these are features of the modern world happening to someone, somewhere, right now.

In the past decade alone Cyprus seized bank deposits during its financial crisis and Greece imposed capital controls preventing citizens from withdrawing their own money. Lebanon's banking system collapsed, trapping savings behind withdrawal limits that have lasted years. Argentina cycled through currency crises with depressing regularity. Nigeria restricted access to foreign currency. China tightened capital flight controls.

There's a consistent pattern of governments reaching for financial controls when faced with fiscal pressure. The national economy and central banks allow bank accounts to be frozen, withdrawals limited, transfers blocked, and assets seized. Thus, the question becomes which assets are seizable and which are not.

Gold has historically served as a hedge against scenarios of fiscal unpredictability. It's hard to confiscate at scale, difficult to track, and holds value across regime transitions. Unfortunately, gold has terrible user experience in the modern world, as it must be physically acquired, verified for authenticity, stored securely, and transported though it's high risk. The friction of its user experience limits its utility as a practical store of value for most.

Bitcoin was supposed to be digital gold. Arguably, it is in certain ways, yet its transparency creates a different vulnerability. If every one of your transactions is visible on a public ledger, the state can simply identify your holdings, track your movements, and apply pressure through legal channels. The transparency that makes Bitcoin trustless also makes it targetable.

The Surveillance Ratchet Surveillance capabilities only move in one direction: expansion.

Governments accumulate data, build systems, hire analysts, and develop analysis techniques. Governments can share information across agencies and even across borders. The infrastructure of surveillance, once built, does not get dismantled, but upgraded.

AI is going to dramatically accelerate these advancements. Pattern matching that once required teams of analysts can now be automated. Metadata that once sat in silos can now be correlated at scale. Behavioral analysis that once took months can now happen in real time. The cost of surveillance per person drops toward zero. The only limit is what data exists to be analyzed.

On transparent blockchains, that data is everything. Literally every transaction that you have ever made is permanently preserved, waiting for better analysis tools. The blockchain does not forget, and neither do the adversaries mining it for information.

What you do today will be analyzed with the tools of tomorrow. Transactions that seem anonymous now may be trivially traceable in five years. Patterns that seem hidden in noise today may be obvious signals once the algorithms improve. The decisions that you make in 2026 must account for the state of privacy and analysis in 2030.

The Precedent We Must Remember One of the most effective tools of authoritarian control is mandatory disclosure. It doesn't begin with confiscation, but with the collection of information. Register your religion. Declare your assets. Report your associations. Though these requirements are presented as administrative and bureaucratic, these often preclude something worse.

Once disclosure becomes mandatory, populations can be segmented, and groups can be identified, analyzed, and assessed. Do they follow a religion that we disapprove of? Do they belong to associations that we find threatening? Do they possess assets that we might want? The separation and distinction precedes the persecution, it's once the data exists that the targeted actions become possible.

Authoritarian control has occurred within living memory, and even happened without the scalability advantages that modern technology provides. The Nazis used paper records and filing cabinets. Today, our digital tools make population-scale identification and targeting effortless.

To hold private assets is to refuse these threats, it is to reject the premise that your financial life should be legible to power, it is a stance against a philosophy that has proven catastrophic when implemented.

AI-powered surveillance is ever expanding. The weaponization of legal systems against disfavored groups is increasing. Capital controls are becoming more common as fiscal pressures mount. Confiscation for political reasons is no longer unthinkable in developed democracies.

The security of your wealth should not depend on who wins elections. Your savings should not be one policy change away from seizure. Your financial privacy should not rely on the continued goodwill of institutions that have demonstrated their willingness to bend the rules.

In sum, the macro case for privacy is that bad things have happened, are happening, and will continue to happen. The question is whether you'll be prepared to face them when they arrive at your door.

5.5 The Fork in History

We're at a branching point. The infrastructure of money is being rebuilt. The choices made now will determine what is possible late, and the branches diverge sharply.

Surveillance Money One path leads to total financial visibility, where every transaction is logged, every donation is analyzed, and every purchase builds a profile. This outcome is the trajectory of our current system.

Central bank digital currencies are being piloted across the globe. For example, China's digital yuan is already deployed at scale, the European Central Bank is developing the digital euro, and the Federal Reserve has studied a digital dollar. Importantly, these systems are surveillance-enabling, not privacy-preserving, by design. The objective is increased visibility: who spent what, where, when, and with whom.

Programmable money further extends the logic of fiscal control, introducing expiration dates on currency that force spending, restrictions on what categories of goods can be purchased, social credit systems where financial access depends on behavior scores, and stimulus payments that can only be used at approved vendors. None of this requires conspiracy, it only requires the infrastructure to have been built and the incentives to use it to arise.

Transparent blockchains fulfill the infrastructure component by providing surveillance without the overhead of building CBDCs. Governments do not need to issue digital currency when citizens voluntarily record their transactions on public ledgers. The outcome is the same: a panopticon where economic activity is legible to anyone with the tools to read it.

The path of total financial visibility ends with money as a means of control. It's not an instrument of voluntary coordination, but a device for social management. Spend 'correctly' and you are left alone, spend 'incorrectly' and you are flagged, restricted, and frozen. The freedom to transact becomes a privilege granted to you by Big Brother.

Freedom Money The other leads to money that cannot be surveilled, censored, or controlled. Transactions are private by default, and account balances are visible only to their owners, making economic activity legible to participants and opaque to observers.

It's important to note that this path does not result in anarchy, a state of being without rules. The result of this path is rules, but rules that are enforced by mathematics rather than institutions. It's not possible to double-spend because cryptography prevents it. It's not possible to inflate the supply because the protocol forbids it. It's not possible to forge transactions because you do not have access to the required keys. The rules are embedded in the system itself, enforced by nodes instead of government, and importantly, immune to discretionary override.

In this future, markets function without the distortive effect of observation. Group coordination remains possible without the influence of surveillance. Dissenting organizations remain possible because financial support cannot be traced. Innovation remains possible because experimentation cannot be monitored. Thus, the conditions for progress described in section 5.1 are preserved.

The Encryption Precedent There's still reason to believe that the freedom path is not foreclosed.

In the 1990s, the United States government tried to ban strong encryption. The NSA and FBI argued that encrypted communications would support criminals and terrorists, and pushed for key escrow systems that would provide the government with backdoor access. These federal organizations classified encryption software as munition, making its export illegal.

The cypherpunks opposed and defeated these measures, but encryption spread anyway. Researchers published algorithms. Developers shipped software. The internet adopted TLS. Today, encryption is not merely legal, but mandatory. HTTPS is required for banking, commerce, and communication. The government that once tried to ban encryption now mandates it to protect citizens.

The transition from "encryption is dangerous" to "encryption is required" took about two decades. It's very plausible that private money is going to follow this arc. Today, financial privacy is treated with suspicion: regulators view it as a tool for criminals, and compliance frameworks assume transparency to be the default. The arguments for communication privacy, deemed legitimate and important, also extend to financial privacy: individuals need protection from surveillance, commerce requires confidentiality, and the alternative is a world where control surfaces are everywhere.

Zcash is legal in the United States, it's even traded on regulated exchanges. It has operated for nearly a decade without being banned. This is not an accident. It reflects the same legal and political logic that protected encryption: the right to use cryptographic tools is defensible, and the benefits of privacy extend far beyond those who would abuse it.

The Choice These paths are mutually exclusive, you cannot have both surveillance money and freedom money. You cannot have both financial privacy and universal transaction monitoring. The infrastructure currently being built will determine which world we inhabit.

Choosing to shield your transactions is not just a personal financial decision, it's a vote for which future path to choose. Your choices reveal your preferences, as every transaction in the shielded pool strengthens the network and every user who adopts private money makes it more viable. The technology exists, now we must decide to use it.

Surveillance money leads to a future where economic freedom is a permission to be granted by the powerful. Freedom money leads to a world where economic freedom is fundamental and guaranteed by mathematics. Which do you choose?

Figure 6: Depositors queue outside Northern Rock, September 2007. The first British bank run in 150 years.

6. Evolution & Economics

6.1 Protocol Generations

Zcash has upgraded its core cryptography twice since launching, and with each generation came better performance, stronger security, and fewer trust assumptions. The protocol of today is substantially better than the protocol of 2016.

Sprout (2016) The original shielded pool proved that private cryptocurrency was possible, as, for the first time, a production network offered cryptographic privacy backed by zero-knowledge proofs.

Sprout was just a prototype clothed as production. Creating a shielded transaction required about 40 seconds of computation and several gigabytes of RAM. Sprout was not usable on phones, and barely usable on laptops. Most transactions remained transparent simply because shielding was too costly.

Sprout also required a trusted setup ceremony, where six participants generated the initial parameters, each taking elaborate precautions to destroy their secret contributions. The ceremony worked, but it left an uncomfortable question: what if someone secretly kept the toxic waste?

Sapling (2018) Two years later, Sapling replaced Sprout's cryptography with something far more efficient. The time required for proof generation dropped from forty seconds to just a few. Memory requirements fell to a few dozen megabytes, and shielded transactions became practical on mobile devices for the first time.

Sapling also introduced features that made privacy more usable. For example, viewing keys let users share read access to their transaction history without exposing spending authority, and diversified addresses let a single wallet generate billions of unlinkable receiving addresses.

Importantly, the trusted setup remained. A new ceremony called Powers of Tau involved hundreds of participants over several months, followed by a Sapling-specific phase. The larger ceremony increased confidence, but the trust model was the same: believe that at least one participant was honest.

Orchard (2022) Orchard replaced the entire system of proofs. Built on the Halo 2 proving system, it didn't require a trusted setup and there was no ceremony. Thus, there's no toxic waste and no trust assumptions about events that happened years ago.

The performance of Orchard is comparable to Sapling, but with slightly larger proofs and no setup requirements. The cryptography is also structured differently, using a new curve cycle (Pallas and Vesta) designed specifically for recursive proofs.

Orchard is the pool that Zcash was always meant to have. The earlier generations were the best technology available at the time; Orchard is what became possible once the research caught up with the vision.

Today Orchard is now the default for new shielded transactions. Some wallets, like Zashi, route users to Orchard automatically and auto-shield transparent funds before spending.

Sapling remains supported but is being phased out. It served its purpose as a bridge between the prototype and the production-ready system, but Orchard is the final destination.

Sprout has deprecated, though the pool still exists on-chain, wallets no longer create new Sprout transactions and users with funds in Sprout are encouraged to migrate.

6.2 Turnstiles

Privacy creates an auditing problem. On a transparent chain, you can count every coin. The supply is the sum of all balances and visible to anyone. If a bug allowed coins to be created from nothing, you would see the total increase.

Shielded pools hide balances. You cannot simply add up what everyone holds because you cannot see what anyone holds. So, if counterfeit coins entered the shielded pool, how would you know?

The answer is turnstiles.

The Mechanism Shielded pools each have their own turnstile, that is, a running tally of the ZEC that has entered and exited the pool. When coins move from the transparent pool into a shielded pool, the turnstile records the deposit, and when coins move back out, it records the withdrawal.

The math is simple. If the turnstile shows 1 million ZEC have entered a pool and 800,000 ZEC have exited, then at most 200,000 ZEC remain. If someone tries to withdraw 300,000 ZEC, something is wrong, either the cryptography failed, or someone is attempting to commit fraud.

Turnstiles do not prevent counterfeiting, instead, they detect it. More precisely, turnstiles detect any attempt to cash out counterfeit coins. You can forge ZEC inside a shielded pool (if you somehow break the complex cryptography), but you cannot spend those coins in the transparent pool without the discrepancy being noted.

The Sprout Bug In 2018, a vulnerability was discovered in the Sprout cryptography. A flaw in the proof system that could have allowed an attacker to create coins without detection inside the shielded pool.

The bug was found by the Zcash team during a security audit and patched before any exploitation occurred, but the episode demonstrated the importance of turnstiles.

If an attacker had exploited the bug, they could have minted arbitrary ZEC within Sprout. but they could not have extracted those coins silently. The moment they tried to move forged ZEC into the transparent pool or a different shielded pool, the turnstile math would break and auditors would see that more ZEC exited Sprout than had ever entered.

Turnstiles would successfully limit the blast radius of any attacks, as even a catastrophic cryptographic failure would not produce undetectable inflation. The damage would be bounded by the pool's capacity, and any attempt to realize the counterfeit value would raise alarms.

6.3 Monetary Policy

Zcash's monetary policy mirrors Bitcoin's, so if you understand one, you understand the other.

Fixed Supply The total Zcash supply is capped at 21 million ZEC. The number was chosen deliberately to match Bitcoin, and no protocol change can increase it. The scarcity is absolute and verifiable.

New ZEC enters circulation through mining rewards. Miners who find valid blocks receive newly minted coins. This issuance follows a predetermined schedule that decreases over time until it reaches zero.

Halving Schedule Block rewards halve approximately every four years. It's approximate because the schedule is defined by block height, not calendar time, but the effect is predictable regardless.

At launch, in 2016, the block reward was 12.5 ZEC (after an initial slow-start period). The first halving, in November 2020, reduced it to 6.25 ZEC. The second halving, in November 2024, reduced it to 3.125 ZEC. The process will continue until the reward becomes negligible and eventually zeroes around the year 2140.

Zcash blocks arrive every 75 seconds, as compared to Bitcoin blocks which arrive every 10 minutes. Zcash releases more blocks per day, but the halving schedule adjusts so that the calendar timing is similar and both networks halve roughly every four years despite their different block rates.

6.4 Funding Development

Zcash made a controversial choice when it launched: to pursue protocol-level funding for development. Rather than relying on donations or corporate sponsorship, a portion of every block reward goes directly to development organizations.

Founders' Reward (2016-2020) For the first four years, 20% of all block rewards went to founders, early investors, employees, and the Zcash Foundation, through what was called the Founders' Reward.

It remained a controversial decision despite the fact that the arrangement was disclosed before launch, and anyone mining or buying ZEC knew the terms. On one hand, the critics saw it as a tax on miners and a windfall for insiders. On the other hand, the supporters saw it as necessary funding for a project that required years of ongoing cryptographic research.

The Founders' Reward ended upon the first halving in November 2020 and every recipient received exactly what was promised. The founders now no longer receive protocol rewards.

Dev Fund (2020-2024) Before the Founders' Reward expired, the community debated what should come next. The result was the Dev Fund, a continuation of the 20% allocation under a different structure.

The new distribution directed 7% of block rewards to the Electric Coin Company (the primary development team), 5% to the Zcash Foundation (infrastructure and governance), and 8% to community grants administered by an independent committee. The founders and early investors were removed from the funding stream.

The Dev Fund arrangement ran between the first halving and second halving in November 2024.

Extended Dev Fund (2024-2025) As the second halving approached, the community voted on the allotment again, and decided to extend the Dev Fund with some modifications.

Development funding continues at 20% of block rewards, but now a portion flows to a "lockbox" controlled by future governance mechanisms rather than existing organizations. The intent is to decentralize funding decisions over time, giving token holders more direct influence over how development money is spent.

6.5 Decentralized Governance

No single entity controls Zcash. Development, infrastructure, and governance are distributed across independent organizations with different jurisdictions, funding sources, and mandates.

The Organizations Electric Coin Company (ECC) is the primary protocol development team. The ECC team maintains the reference node implementation, develops the Zashi wallet, and drives the core research. ECC is a subsidiary of the Bootstrap Project, a 501(c)(3) nonprofit based in the United States.

Zcash Foundation handles infrastructure, community programs, and grants. The foundation's team developed Zebra, an independent node implementation written in Rust, ensuring the network does not depend on a single codebase. The Zcash Foundation is a 501(c)(3) public charity, also US-based but operationally independent from ECC.

Shielded Labs focuses on long-term research and ecosystem development. Based in Switzerland and funded by donations rather than protocol rewards, it provides geographic and structural diversity to the contributor base.

Tachyon, led by cryptographer Sean Bowe, is building the infrastructure for Zcash to scale. Bowe was the architect behind Halo 2 and much of Zcash's core cryptography. The Tachyon project aims to enable global private transactions through innovations in how wallets sync with the network without leaking information to servers.

These four organizations collaborate but are not required to answer to each other. They can disagree and they sometimes do. The diversity of aims and perspectives is a feature that prevents capture and ensures multiple perspectives inform protocol decisions.

The ZIP Process Protocol changes follow the Zcash Improvement Proposal (ZIP) process, meaning that anyone can propose a change. Proposals are debated publicly, refined through feedback, and accepted or rejected based on technical merit and community consensus.

Major decisions skip the ZIP process and are resolved through community-wide polling. The Dev Fund extensions in 2020 and 2024 both involved extensive public deliberation and sentiment gathering before implementation. Input was taken from token holders, miners, and community members.

Figure 7: The Enigma machine, used by Nazi Germany to encrypt military communications during World War II. Operators changed settings daily, producing messages that appeared as random gibberish to interceptors.

7. Zcash VS . . .

Privacy comes from value at rest, not value in motion.

This single principle explains why most privacy solutions fail and why base layer encryption is the only architecture that works. Once you understand it, the landscape of privacy technologies becomes clear.

7.1 Tornado Cash and Mixers

Consider what happens when you use a mixer. You deposit funds, wait for some period, then withdraw it to a fresh address. The goal is to break the link between your input and output, but both the deposit and the withdrawal are visible. An observer sees when funds enter and funds exit. The mixer tries to obscure which input corresponds to which output, but this does not ensure privacy.

Rather, it's adding privacy to value in motion, which fails for the fundamental reason that the entry and exit points leak information.

The deposit reveals the time and amount. The withdrawal reveals the time and the amount. If these correlate, privacy is broken. If you deposit 1.5 ETH and someone withdraws 1.5 ETH an hour later, the connection is obvious. Mixers try to solve this with fixed denominations and delays, but the leakage of information remains. Correlations are bound to emerge given enough data and sophisticated analysis.

AI worsens these risks. Pattern matching that was once impractical becomes trivial to solve with timing analysis, amount clustering, and behavioral patterns. Every mixer is a puzzle waiting for algorithms to develop ways to solve it.

The only way to truly separate incoming and outgoing transactions is to separate them on the aspects of both time and value. The incoming transaction must not cause the outgoing transaction. The amounts and the transaction timing must be unrelated.

The private system therefore serves as an actual store of value. Money goes in, it sits, time passes, life happens, until eventually, unrelated amounts go out for unrelated reasons. The deposit and withdrawal are not two parts of a single operation, they are independent events separated by months or years of genuine storage.

You could, in theory, do this with a mixer like Tornado Cash and leave funds in the pool indefinitely, but this is impractical because you cannot do anything with those funds while they sit there.

Additionally, Tornado has fixed denomination pools, so you cannot send arbitrary amounts within the pool, you cannot transfer from one Tornado position to another, nor can you use it to pay someone or to interact with any application. To use your funds for anything at all, you must withdraw to a transparent Ethereum address, re-exposing yourself to the surveillance layer.

Zcash is different. Shielded-to-shielded transfers are native, so you can receive funds, hold them, spend arbitrary amounts, receive change, and transact again, all without ever touching the transparent layer. Bridge from the shielded pool to other chains using Near Intents, paying in shielded ZEC while the recipient receives whatever asset they want. The shielded pool is not a waiting room, it's a fully functional monetary system.

This is the architectural distinction that matters. Mixers are escape hatches from transparent systems - you visit them, you wait, you leave. Zcash's shielded pool is a destination - you can live there.

7.2 Monero

Monero is the most widely used privacy cryptocurrency besides Zcash. It represents a fundamentally different approach to the information leakage problem, and in understanding why their approach fails, we will clarify why Zcash's approach works.

Monero's approach is to use ring signatures. When you spend funds, your transaction includes your real input plus 15 decoys sampled from the blockchain. An observer sees 16 possible senders and cannot tell which one is real.

This sounds robust. Sixteen possibilities per transaction. The real spend is hidden among many fakes. In reality, this just means that the privacy is probabilistic, but not cryptographic.

Law enforcement agencies have successfully traced Monero transactions. There's even a documented case of Japanese police analyzing Monero transactions to identify and arrest eighteen suspected fraudsters.

The fundamental issue is the anonymity set. Each Monero transaction hides among 16 outputs, whereas each Zcash shielded transaction hides among every note ever created in the pool. There are millions of these notes, so the privacy Zcash offers is not incrementally superior, but categorically superior.

Sixteen is a small number, definitely small enough to attack probabilistically, especially now that timing analysis, amount patterns, and behavioral heuristics can narrow the candidates. Sixteen is small enough that sufficient compute and data will eventually crack it.

There is no probabilistic attack that works against a privacy set of millions of notes. It's impossible to narrow the candidates through elimination because nothing is eliminated. The note you spent remains indistinguishable from millions of others forever.

Monero's developers understand this limitation, and there is active research into replacing ring signatures with zero-knowledge proofs, effectively, it's planning on adopting Zcash's approach; an implicit acknowledgment that decoy-based privacy has a ceiling.

The distinction is simple: Monero obfuscates, Zcash encrypts. Obfuscation degrades over time as analysis techniques improve, encryption does not.

On top of the technical weaknesses, Monero also carries cultural baggage. The community has embraced an association with illicit use, making its institutional adoption nearly impossible. This is part of why Monero has been delisted from virtually every major exchange while Zcash remains available on Coinbase, Gemini, and others. Privacy technology needs a path to legitimacy, and Monero has made that path harder for itself than it needed to be.

7.3 Privacy Pools

Privacy Pools present a different approach to privacy solutions. Rather than hiding among random decoys or encrypting everything, it lets users prove they are not associated with known bad actors. You can withdraw from a pool while demonstrating your funds did not come from sanctioned addresses or flagged transactions.

The design is clever, association sets let you define who you are willing to be grouped with. Thus, you prove membership in a "clean" set without revealing which specific deposit is yours. Regulators receive assurance that funds are not tainted and users receive some privacy, everyone is happy.

Except, this inverts due process.

The premise of Privacy Pools is that you must prove your innocence. It's on you to prove that your funds are not associated with criminals and to opt into a set of "good" users and provide cryptographic evidence that you belong there. The default assumption is suspicion, and the burden falls on you to clear it.

In functioning legal systems, you do not have to prove that you are not a criminal - the prosecution must prove that you are. Privacy Pools normalize the opposite, that you are guilty until you prove yourself innocent through the approved association sets.

The implications abound, as your privacy depends on what others choose to disclose. If members of your association set start proving exclusion from various activities to clear their own names, the remaining members become more suspicious. There is constant pressure to prove more, disclose more, and further narrow your set. The system creates chilling effects by design.

There does not exist "compliant encryption" for messaging. Signal does not ask you to prove that you are not conversing with terrorists, it accepts that communication privacy is a right, at the cost of benefiting criminals.

There is no reason that it should be any different for finance. The argument that money is special or that financial privacy uniquely enables harm do not survive scrutiny. Criminals use cars, phones, and the internet, yet, we do not require proof of innocence to drive, call, or browse.

Privacy Pools attempts to find the middle between surveillance and freedom. It offers privacy that is conditional on compliance, requires proving you deserve it, and that can be withdrawn if you fail to convince others of your innocence.

It's not privacy, it's permissioned finance with extra steps.

7.4 Aztec and Private L2s

Ethereum Layer 2s, when supplemented with privacy features, represent serious engineering. Projects like Aztec are building encrypted rollups with sophisticated cryptography. The technology is sound and the team is talented; this is not a critique of their technical capabilities.

Fundamentally, Aztec and Zcash are solving different problems.

Aztec is a smart contract platform. Its value proposition is private programmability: encrypted DeFi, confidential computation, and private applications. This is valuable as it enables use cases not addressed by Zcash. If you want to interact with complex financial protocols without exposing your positions, use an encrypted smart contract chain.

Zcash is money. Its value proposition is being a private store of value and medium of exchange. The memetics are clear: it's essentially encrypted Bitcoin. A place to hold wealth privately, for years or decades, with confidence that the system will remain in existence and continue to function.

These are not the same use case, therefore, their requirements differ.

A store of value needs to be Lindy. It needs to sustain years of operation under adversarial conditions, survive market cycles, regulatory pressure, and technical challenges without breaking. Zcash has already built nearly a decade of this history. Aztec is new, and though the cryptography may be perfect, the system has not been tested over time. This may be acceptable for experimental applications, but not for the security of wealth holdings.

A store of value also needs memetic strength. Bitcoin succeeded partly because "digital gold" is a powerful narrative that people understand and believe. "Encrypted Bitcoin" gives Zcash a similar anchor, inheriting Bitcoin's monetary properties while adding the privacy that Bitcoin lacks. Aztec does not have this narrative, it's simply a privacy infrastructure layer, not a monetary network.

Beyond the technical design, there is a social layer.

Zcash's community formed around a shared commitment to privacy as a non-negotiable principle, and over nearly a decade it has resisted legal, political, and reputational pressure to weaken that commitment. By contrast, a Layer-2 system inherits its ultimate norms and governance constraints from its Layer-1. In Ethereum's case, it is unclear whether the broader community would consistently defend strong encryption and transaction privacy if faced with regulatory pressure. For an asset intended to function as a long-term store of value, that uncertainty itself constitutes risk.

Aztec and similar projects will likely find significant demand for private applications, but for the core use case of private money, a place where wealth can rest indefinitely, they serve a different purpose than Zcash.

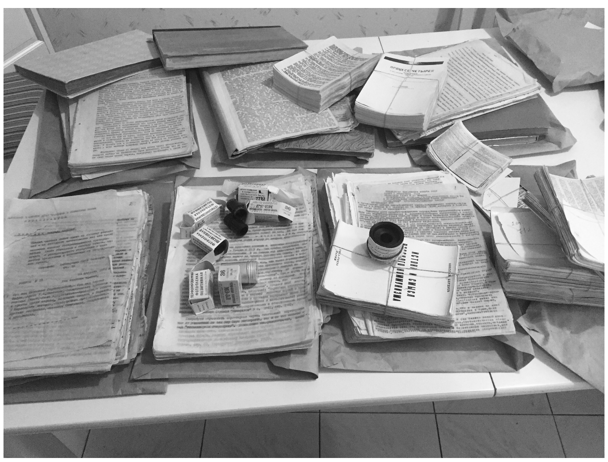

Figure 8: Samizdat — Soviet citizens copying and distributing banned literature by hand to evade state censorship. Possession meant prison.

8. Misconceptions

8.1 "Zcash Is Not Private by Default"

This misconception confuses what has historically been the default for wallets with protocol design.

New ZEC is always minted into the shielded pool, meaning that every new coin enters into existence as encrypted. The protocol is private by default at the most fundamental level: issuance.

The misconception about default privacy arose because early wallets defaulted to transparent addresses for reasons of practicality. In Sprout and Sapling, shielded transactions were computationally expensive and exchanges required transparent deposits. So, the path of least resistance was often transparent.

Orchard has now made shielded transactions more efficient and wallets like Zashi enforce shielding by default, automatically moving any transparent funds into the shielded pool before allowing you to spend. The user experience has become private-first.

The transparent option remains for specific use cases, such as exchange compatibility, regulatory compliance, and user choice, but the default path through modern Zcash is shielded from start to finish.

8.2 "The Anonymity Set Is Small"

This misconception stems from confusing Zcash with decoy-based systems.

As we covered above, in Monero, your transaction hides among a fixed number of decoys. If there are 16 possible senders, your anonymity set is 16. Therefore, many critics assume that Zcash works similarly: if few people use the shielded pool, then your transaction hides among just a few others.

However, this is wrong. Zcash does not sample decoys, it uses Merkle tree membership proofs.

When you spend a shielded note, you prove that it exists somewhere in the commitment tree that contains every note ever created, without revealing which note it is. The verifier learns only that your note is one of the millions, not hundreds or thousands, in the tree.

The Orchard pool contains over 5 million notes, that's the anonymity set for every shielded transaction and it grows with every transaction and never shrinks.

The size of the transparent pool is irrelevant, even if 99% of ZEC sat in transparent addresses, the shielded 1% would have an anonymity set of every shielded note ever created. The two pools are mathematically independent.

8.3 "Optional Transparency Weakens Privacy"

This misconception assumes the transparent pool somehow contaminates the shielded pool.

The two are independent systems, transparent ZEC and shielded ZEC operate in parallel. Transactions on the transparent side reveal nothing about the shielded side. The cryptographic guarantees of shielded transactions do not depend on how much ZEC sits in transparent addresses.

Think of it as two separate ledgers that happen to share a currency; activity on one does not affect the privacy properties of the other.

The transparent option exists because it provides real value. Exchanges can use transparent addresses for deposits and withdrawals, satisfying compliance requirements while still listing ZEC. This permits users who need auditability to choose it and applications that require transparency to build on it.

The optional transparency does not compromise shielded privacy, it simply increases Zcash's adoptability, something that completely private-by-default chains lack. This is exemplified by the fact that Monero has been delisted from major exchanges, while Zash remains on Coinbase and Gemini.

8.4 "Zcash Uses a Trusted Setup"

This misconception has failed to update from what once was true, but isn't anymore.

Sprout and Sapling required trusted setup ceremonies where participants generated cryptographic parameters and destroyed the secret values used to create them. If anyone kept those secrets, they could forge proofs and mint counterfeit ZEC.

As covered above, the ceremonies were elaborate, consisting of multiple participants, air-gapped machines, and even subsequently destroyed hardware. Despite these strong precautions, the trust model introduced modicums of doubt.

Orchard resolved this issue by using Halo 2, a proving system that requires no trusted setup. There was no ceremony, no toxic waste, and no threat of secrets not being destroyed. Now, the parameters come from public, verifiable data.

Zcash's shielded pool is now trustless, just like Bitcoin, and the security is guaranteed by cryptographic mathematics, rather than faith in ceremony participants.

8.5 "There Was a Premine"

This misconception is fundamentally incorrect. No coins existed before the genesis block, there was zero premine.

The fallacy arises from the Founders' Reward, because during Zcash's first four years, 20% of block rewards went to founders, investors, employees, and the Zcash Foundation. However, this was not a premine, it was simply a portion of ongoing issuance, created through mining, just as for every other coin.

This distinction matters. Premine would have created coins before anyone else can participate, whereas the Founders' Reward created coins at the same rate as miner rewards, and then directed them differently. Miners received 80% of each block, and founders received the remaining 20%, importantly, both came from the same issuance schedule.

The terms of the Founder's Reward were fully disclosed before launch, both the whitepaper and the website explained its reason and its process. So anyone mining or buying ZEC in 2016 knew exactly how its distribution functioned, and there was no hidden allocation, secret stash, or coins that appeared from nowhere.

The Founders' Reward ended upon the first halving in November 2020, at that point, every recipient had received what was publicly promised and nothing more.

8.6 "Devs Get 20% of Mining Rewards"

This misconception conflates two programs and their respective recipients.

The Founders' Reward ran from 2016 to 2020, directed 20% of block rewards to founders, early investors, employees, and the Zcash Foundation, and ended at the first halving. Therefore, founders haven't received protocol rewards since 2020.

The Dev Fund replaced the Founder's Reward and ran from 2020 to 2024. The Dev Fund also allocates 20% of block rewards, but to different recipients. ECC receives 7% for protocol development, the Zcash Foundation receives 5% for infrastructure and grants, and community grants, administered by an independent community, receive 8%.

Contrary to misconceptions, Dev Fund does not serve to support personal enrichment. Rather, it funds organizations that employ developers, maintain infrastructure, and award grants to ecosystem projects. The fund pays for Zcash's continuous improvement.

The alternative is Bitcoin's model, which relies on donations and corporate sponsorships—an approach with its own tradeoffs. Zcash instead adopted protocol-level funding to support sustainable development, and nearly nine years of continuous upgrades suggest that this choice has been justified.

8.7 "The Zcash Foundation Controls Zcash"

This misconception fails to understand that no single entity controls Zcash.

In fact, there are four independent organizations that contribute to the protocol. The Electric Coin Company (ECC) builds the reference implementation and Zashi wallet. The Zcash Foundation maintains Zebra, an independent node implementation, and administers grants. Shielded Labs conducts research from Switzerland. The Tachyon team, led by Sean Bowe, builds scalability infrastructure.

These organizations operate in different jurisdictions, with different funding sources, and different mandates. Though they collaborate on protocol development, they can disagree on issues and do not answer to a common authority.

The separation of these organizations was implemented deliberately. In case one of the organizations is pressured, captured, or compromised, the others can continue to function and maintain the system. The protocol does not depend on a single team, and the two independent node implementations mean that there's no single authoritative codebase.

Zcash is more decentralized in governance than most cryptocurrency projects. Arguably, it's more decentralized than Bitcoin, as the latter is dominated by a single implementation mechanism and a handful of maintainers control what gets merged.

8.8 "The Mossad Is Behind Zcash"

This misconception is simply conspiracy theory, there is no evidence to support it.

The conspiracy either points to the fact that some founders have connections to Israel or the fact that academic cryptographers are involved in the project. Based on this logic, any technology developed in part by people with ties to any country is controlled by that country's intelligence services.

Zcash is open source, literally every line of code is public and auditable; its cryptography is published mathematics, peer-reviewed and scrutinized by researchers worldwide. If there were a backdoor, it would be visible in the code and the proofs.

Additionally, the four independent organizations, based in multiple countries, contribute to the protocol. The community includes developers, researchers, and users from every continent. It's simply irrational to believe that an intelligence agency controls a globally distributed open source project due to the descent of any of the early contributors.

The same conspiracy thinking could be used to target any technology. Signal was developed in part from grants from the United-States government, does that mean that the CIA is behind Singal? Linux has contributors from every major government and corporation, does that mean that multiple governments have compromised it?

The code is open source and the math is public, both invalidate the conspiracy.

8.9 "Criminals Use Monero for a Reason"

This misconception implies that criminals would have necessarily identified the strongest privacy technology in order to conceal their crimes, but this gives criminals too much credit.

Criminals are not cryptographers. They do not evaluate elliptic curve implementations or compare anonymity set constructions. Instead, they use what's familiar and what already has a reputation in their communities.

Monero built its brand around being the 'crime coin' and therefore attracted criminals. This demonstrates a pattern of reinforcement between Monero's brand and criminals' use of Monero, not Monero's technical superiority.

The comparison of Monero and Zcash's privacy capacities favours Zcash. Monero hides transactions among 16 decoys, while Zcash hides notes among more than 5 million others. Monero's decoys can be eliminated through chain analysis with time, while Zcash's cryptographic indistinguishability makes such chain analysis decryption impossible. Monero cannot be criminals' choice for privacy reasons when law enforcement has successfully traced Monero transactions, as exemplified by the Japanese case covered above.

Criminals also use cash, prepaid phones, and even standard email in their business, but no one argues these are used because they're the most secure options available. Rather, these means are used because they are the most accessible and familiar options.

The criminals' choices reveal decisions based on marketing and network effects, not reasoned decisions based on cryptographic strength.

8.10 "Monero Is More Private Because All Transactions Are Private"

This misconception argues that Monero's mandatory privacy somehow means that it's more secure than Zcash's optional privacy model. The confusion arises from failing to distinguish design defaults from cryptographic strength.

As covered above, Zcash is also private by default, as ZEC is minted into the shielded pool and modern wallets enforce shielding. The default path is fully encrypted.

Even if Monero and Zcash's default paths differed, the distinction would not determine their privacy strengths.

The mechanism matters more than the setting.

Monero's mechanism: ring signatures with 16 decoys, hiding your transaction among 16 possible senders. As the decoys can be eliminated over time through chain analysis, the anonymity set shrinks retroactively and the connection can be traced.

Zcash's mechanism: zero-knowledge proofs over a Merkle tree of over 5 million notes. Your transaction could have spent any of the notes, and there is no process of elimination to trace the origin. The set only grows and the cryptographic indistinguishability is permanent.

The default of weak locks on every door is not preferential to the default of a strong lock on the doors that matter, and the option to add locks to the others. Mandatory weak privacy is simply weak privacy, and optional strong privacy is simply strong privacy.

The correct question is not whether privacy is the default, but whether the privacy holds up under adversarial analysis. Zcash's privacy does, Monero's privacy does not.

Figure 9: Zcash team in the early days, featuring among others Zooko Wilcox-O'Hearn, co-founder of Zcash, and Jay Graber, then a junior developer in the Zcash team and who went on to later become CEO of Bluesky.

9. Road Ahead

9.1 Project Tachyon

Tachyon addresses three scaling bottlenecks in Zcash: double-spend prevention, blockchain scanning, and transaction size. Double-spend prevention is the hardest of the three, and its solution reveals what makes Tachyon a genuine breakthrough rather than another incremental optimization.

The Nullifier Problem Zcash prevents double-spending through nullifiers - when you spend a note, you reveal a nullifier: a random-looking string that functions as a revocation token. Nullifiers can't be linked to the notes that they revoke, but if you try to spend the same note twice, you reveal the same nullifier and the network knows to reject the duplicate.

The nullifier problem is that every validating node must store every nullifier ever revealed, forever. It's unsafe to prune old nullifiers because someone may decide to respend an old note. At one hundred transactions per second, this would create roughly one gigabyte of state growth per day. If you're not familiar, that's an extreme amount compared to most blockchains, including high-throughput chains like Solana.

Why Naive Solutions Fail The cryptographic community has known for years that recursive proofs could solve this problem. Rather than the network tracking nullifiers, recursive proofs would permit users to prove they haven't double-spent. Attach the proof to the transaction and validators verify the proof and then prune old nullifiers.

The devil's in the details.

- **Approach 1:** Download the full chain history to your wallet and construct the proof locally. This works cryptographically but fails practically, as your wallet bears the bandwidth and computational cost of every transaction everyone else makes and phones can't do this.
- **Approach 2:** Add an intermediary service. Send your transaction to the service, let it construct the proof using full chain history, and then broadcast it. This works, but it introduces massive latency. The service must process the entire chain for every transaction, and requires you to trust the service with your transaction data.
- **Approach 3:** Send your nullifiers to the service in advance, receive the proofs back, then later, attach those proofs to your transactions and broadcast. This may seem clever, but it has a fatal flaw: the service is able to observe which nullifiers you're preparing to spend, and can therefore link your transactions together, leaking your primacy to the intermediary.

Oblivious Synchronization Here's Tachyon's solution: A service that proves that you haven't double-spent by performing the computation, without seeing what appears in the final transaction and learning which nullifiers you're spending. The service cannot distinguish your transactions from anyone else's.

Technically, this is defined as being an "oblivious" service. The service is blind to the actual data that it's processing on your behalf, so you get the computational help without trusting the helper.

The result is validators that don't store the full nullifier history. Therefore, users aren't exposed to costs that scale with total network activity, and ledger indistinguishability, Zcash's core privacy property, remains intact.

The Other Bottlenecks Blockchain scanning, the process of identifying which transactions belong to you, is solved through protocol design changes rather than new cryptography. The current requirement to trial-decrypt every transaction becomes replaced with a more efficient payment protocol.

Transaction size and verification time use the same recursive proof techniques. The marginal transaction size and the verification time drop to about the scale of Bitcoin. Thus, a fully private Zcash transaction ends up being around the same size and speed as a transparent Bitcoin transaction.

What This Enables Once Tachyon is implemented, Zcash's scaling constraints will become the same as those facing other blockchains - bandwidth and latency. The cryptographic overhead that made privacy expensive disappears and even a phone can transact privately without processing the full chain. A node can validate without storing gigabytes of nullifier state.

The tradeoff between privacy and scale, long assumed to be fundamental to encrypted money, turns out to be an engineering problem with a cryptographic solution.

9.2 Network Sustainability Mechanism (NSM)

Bitcoin faces a looming problem: As block rewards halve towards zero, transaction fees must compensate miners for retaining the network's security. Whether fees will suffice remains an open question, but the anticipated alternatives, such as tail emissions, are going to break the 21 million cap.

Zcash inherits this problem, but the Network Sustainability Mechanism solves it without breaking the cap.

The Mechanism The NSM allows ZEC to be burned from circulating supply and reintroduced as future block rewards. Burning 1 ZEC now causes 0.5 additional ZEC to be issued over the next four years, 0.25 over the following four years, and so on. The issuance follows an exponential decay model approximating the existing four-year halving schedule.

In the short-term, this results in reduced circulating supply and increased scarcity. In the long-term, there will be more ZEC available for block rewards further along the emission curve, sustaining miner incentives without exceeding the 21 million cap.

Three ZIPs ZIP 233 establishes voluntary burning, meaning that users can donate directly to the Zcash network rather than to organizations or individuals. Wallets could offer an option to burn ZEC when transacting. Verifiable burns enable token-gated communities or identity badges that prove contribution to network sustainability.

ZIP 234 smooths the issuance curve, so that instead of abrupt halvings, emissions decay continuously. This provides a predictable mechanism for reintroducing burned coins without sudden supply shocks.

ZIP 235 burns 60% of transaction fees. Currently, this amounts to roughly 210 ZEC per year, which is a negligible amount. The point is to establish the mechanism while fees are low and miners have no economic incentive to oppose it, future fee structures remain a community decision to be made once NSM is operational.

Future Applications The NSM creates infrastructure for use cases that the community will encounter down the line:

- **ZSA fees:** Minting, transacting, or bridging Zcash Shielded Assets could burn a portion to compensate ZEC holders.
- **Legacy support fees:** Users storing funds in older pools could pay fees, thus incentivizing migration to newer, more secure pools.
- **Privacy incentivization fees:** Transparent address usage could incur fees to compensate for the reduced anonymity set.
- **EIP-1559 style dynamic fees:** Base fees adjusted with network congestion, partially burned.

Why Now? Currently, Transaction fees are minimal, so implementing the burn mechanism now avoids the political difficulty that Ethereum faced with EIP-1559, where miners had strong incentives to oppose fee burning. If implemented now, the precedent will exist by the time Zcash fees become significant.

The NSM can continue Zcash's tradition of improving on Bitcoin's design. Privacy and the Dev Fund already differentiate Zcash, and this upgrade would add a third differentiation: a mechanism for long-term network sustainability that's not present in Bitcoin.

9.3 Quantum Resistance

Zcash's relationship with quantum computing is more nuanced than most other cryptocurrencies. The protocol already provides significant post-quantum privacy protections in common scenarios, as a result of deliberate design choices made since the project's inception.

What's Already Protected Quantum adversaries cannot compromise onchain anonymity. Zcash's nullifiers, the mechanism preventing double-spends, use keyed pseudorandom functions built on symmetric cryptography, and these primitives remain secure against quantum attacks. The commitment schemes are therefore perfectly hidden, and the symmetric encryption uses key sizes designed for post-quantum security.

This contrasts sharply with other privacy cryptocurrencies. Monero's key images, the equivalent of nullifiers, would become transparent to a quantum adversary, and the transaction graph would be revealed. Zcash's construction avoids this vulnerability entirely.

The Two Threats Quantum computers threaten two distinct properties: privacy and soundness.

Privacy concerns center on "harvest now, decrypt later" style attacks. An adversary could collect encrypted transaction data today and decrypt it later, once quantum computers have arrived. This primarily affects in-band secret distribution, the mechanism for transmitting transaction details to recipients. Tachyon's design removes in-band secret distribution entirely, protecting against this future threat.

Soundness concerns center on elliptic curve cryptography, which could be broken by quantum computers. Though this would enable counterfeiting or theft, it would not compromise privacy. Therefore, the threats differ in their urgency as privacy breaks are retroactive (past transactions become vulnerable), while soundness breaks often are not (you can react when quantum computers appear).

Quantum Recoverability ECC has developed techniques for "quantum recoverability" in Orchard. After upcoming wallet changes, and assuming that quantum computers appear, users would be able to recover funds through a special mechanism that prevents quantum adversaries from stealing them, a mechanism that also protects privacy.

The timeline for wallet integration is that it's released in 2026, so users that shield their coins and await these improvements will be protected.

Best Practices Today Shield your coins. The shielded pool's design already provides substantial quantum resistance for on-chain privacy. Treat addresses as secrets whenever possible. Turnstiles remain the final defense: even if counterfeiting occurred, it would eventually become detectable when funds exit shielded pools.

Zcash's cryptographers will remain ahead of developments, because the protocol's modular design, which isolates vulnerable primitives, enables future upgrades without overhaul.

Figure 10: "Tank Man" standing in front of a column of tanks near Tiananmen Square in Beijing on June 5, 1989.

10. Conclusion

In conclusion, this article opened with a simple observation: unless you're using cash, every purchase you make is tracked and stored indefinitely. Bitcoin could have fixed this, but it didn't. The blockchain that was supposed to free us from financial surveillance became the most comprehensive surveillance tool ever deployed.

Zcash took a different path. Instead of transparency by default, with privacy bolted on as an afterthought, it first solved the hardest problem: how do you verify transactions without seeing them?

The answer required zero-knowledge proofs, commitments that hide amounts, nullifiers that prevent double-spends without linking transactions, and a note model that shatters the transaction graph entirely. Nine years of protocol evolution followed: Sprout proved that privacy was possible, Sapling made privacy practical, and now Orchard has made it trustless.

The result of this evolution is ledger indistinguishability. Two shielded transactions cannot be told apart, not by observers, validators, or nation-states with unlimited resources. The data is not just obscured or mixed with decoys, it is encrypted. What the network sees is mathematically indistinguishable from random noise.

The road ahead remains demanding. Tachyon removes the bottlenecks that constrain scale. NSM creates sustainable economics. Quantum resistance is a solvable problem with work already underway. The foundation is built. The cryptography works. The privacy is real.

Before us are two futures: One where every transaction is visible, controllable, and reversible by whoever holds power, and another where money is as private as thoughts.

Zcash is how we get the second one.

Further Reading

- The Sovereign Individual by James Dale Davidson
    - My Zcash Investment Thesis by Frank Braun
    - Zcash Protocol Specification by Daira-Emma Hopwood et al.
    - Zcash: A Zero to Hero's Guide by Arjun Khemani
    - Understanding Zcash: A Comprehensive Overview by Youssef Haidar
    - Inside Zcash: Encrypted Money at Planetary Scale by CoinDesk Research
    - The Case for a Small Allocation to ZEC by Sacha
        - Freedom Money by Arjun Khemani
    - Bitcoin Whitepaper by Satoshi Nakamoto