DAHBA $DAH

# Dahba: Fast Proof-of-Work Cash with Optional Privacy

The Dahba project · dahba.cash

Version 0.2 · November 13, 2026


## ABSTRACT

Dahba is a proof-of-work currency that produces a block every tenth of a second and is optionally private: the sender chooses per transaction. It joins two existing systems and invents no cryptography. Consensus is GHOSTDAG [2], the block DAG protocol run by Kaspa, at ten blocks a second. Privacy is the Orchard shielded protocol built for Zcash [4, 5], which hides the sender, the receiver and the amount behind a zero-knowledge proof and needs no trusted setup [6]. A shielded pool was designed for a single chain, and a block DAG makes blocks in parallel. We keep the pool's state as entries in the UTXO set, so the machinery that already orders, merges, reverts, commits and prunes coins does the same for private notes. The reward of a block is paid into the pool, so the chain does not show who mined it. On a test network a transaction of either kind reaches the receiver 0.2 seconds after it is sent. There is no public network yet. This paper describes the software as it stands and lists what is missing.


## CONTENTS


1. Introduction
2. Background
3. Consensus
4. Two kinds of funds
5. The shielded pool in the UTXO set
6. Validation rules
7. Privacy model
8. Proof of work
9. Supply and incentive
10. Performance
11. Pruning and synchronisation
12. Programmability
13. Limitations and open work
14. Conclusion

# 1 Introduction

Bitcoin [1] showed that a network with no central party can agree on the order of transactions by proof of work. It pays for that agreement with time: one block about every ten minutes, and every transaction open for anyone to read. The coins that added privacy kept the single chain and most of its wait. Zcash produces a block every 75 seconds and Monero one every two minutes. Kaspa removed the wait by replacing the chain with a block DAG, and confirms transactions in about a second, but every transaction on it is public.

Dahba puts the two properties in one proof-of-work system: the block rate of a block DAG, and a shielded pool for transactions that should stay private. Each part is reused from an existing, public project.

| PART | TAKEN FROM | LICENCE |
|---|---|---|
| Node, GHOSTDAG consensus, pruning, peer-to-peer network, RPC | rusty-kaspa v2.1.0 [8] | ISC |
| Shielded pool: notes, zero-knowledge proofs, keys | the `orchard` library, version 0.16 [9] | MIT / Apache-2.0 |
| Proof verification in script, transaction lanes | already part of rusty-kaspa v2.1.0 | ISC |

What is new is the join between the shielded pool and the DAG (sections 5 and 6), block rewards that are paid into the pool (section 7.1), a change to the last step of the mining function (section 8), and the supply schedule (section 9).

# 2 Background

## 2.1 The chain and its speed limit

In Bitcoin a block names one earlier block, and nodes treat the longest chain as the record. When two nodes find a block at the same moment, the network keeps one. Nodes work on the first they received and save the other, and the tie breaks when the next block is found [1]. The work in the losing block secures nothing. To keep such collisions rare, blocks have to be far apart compared with the time a block needs to cross the network. Hence the long block interval.

## 2.2 Block DAGs and GHOSTDAG

PHANTOM and GHOSTDAG [2] generalise Nakamoto's rule. A block names every tip its miner knew, not one, so the blocks form a directed acyclic graph and blocks found at the same moment are all kept. The protocol then has to turn the graph into one order. PHANTOM does this by finding

the largest set of blocks that are well connected to each other, which is an NP-hard problem. GHOSTDAG is the greedy version that a node can run on every block. Its authors show that the order it produces becomes irreversible except with a probability that shrinks exponentially [2]. Section 3 gives the rule as Dahba runs it.

### 2.3 Shielded transactions

Zerocash [3] defined a decentralised anonymous payment scheme: a ledger on which a transaction hides its origin, its destination and its amount, while every node can still check that no coin was created or spent twice. Zcash is the implementation of that scheme [4]. Funds are held in **notes**. The ledger stores a commitment to each note in a Merkle tree. Spending a note reveals its **nullifier**, a value that only the owner can compute and that cannot be linked to the commitment, together with a zero-knowledge proof that the nullifier belongs to some note in the tree and that values balance. A nullifier seen twice is a double spend.

Orchard [5] is the current form of this design. It works over the Pallas and Vesta curves, uses the Sinsemilla and Poseidon hash functions, and proves its statements with Halo 2, which needs no trusted setup [5, 6]. Its unit is the **action**, which spends one note and creates one note, either of which may be a dummy, so an observer cannot count real inputs and outputs.

### 2.4 Ring signatures

CryptoNote [7], the design behind Monero, takes a different road. It names two properties. **Untraceability**: for each incoming transaction all possible senders are equally likely. **Unlinkability**: for any two outgoing transactions it is impossible to prove they were sent to the same person. It gets the second from one-time destination keys and the first from a ring signature, in which the real input is signed together with decoy outputs chosen by the sender. A **key image** published with the signature exposes a second spend of the same output. The signature grows with the number of decoys, so the set a sender hides in is the ring he pays for.

In a shielded pool the set is every note ever committed to the tree, and the proof does not grow with it. Dahba uses the shielded pool for that reason. Monero's privacy is mandatory. Dahba's is optional (section 7).

---

## 3 Consensus

Dahba runs GHOSTDAG unchanged from rusty-kaspa [8]. For a block $B$ in the DAG, **past**($B$) is the set of blocks reachable from $B$ by following parent links, **future**($B$) is the set of blocks that reach $B$, and **anticone**($B$) is every other block: those made without knowledge of $B$, and without $B$ knowing

of them. Honest blocks have small anticones, because an honest miner publishes at once and builds on everything it has seen. The anticone of an honest block is bounded by the number of blocks made while it travels.

A set of blocks $S$ is a **k-cluster** if every block in $S$ has at most $k$ blocks of $S$ in its anticone. GHOSTDAG colours blocks **blue** (in the cluster) or **red** (outside it) as follows.

1. Each block takes as its **selected parent** the parent with the most accumulated blue work, and inherits that parent's blue set.
2. The block's **merge set** is everything in its past that is not in the past of its selected parent. These are the parallel blocks it merges.
3. The merge set is visited in a fixed order. A block is coloured blue if adding it keeps the blue set a k-cluster: it has at most $k$ blue blocks in its anticone, and no blue block's anticone grows beyond $k$. Otherwise it is red.
4. The chain of selected parents back from the heaviest tip is the **selected chain**. The order of all blocks follows this chain, each chain block followed by its merge set.

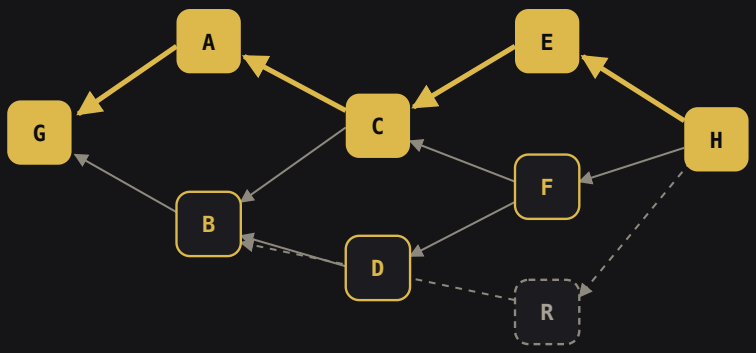



**Figure 1.** A block DAG. Every block points to the tips its miner knew. Filled blocks are the selected chain. Outlined blocks B, D and F were made in parallel and are blue: the chain merges them and their transactions count. R was built on B alone, without the blocks made since. It is red: still ordered, after the blue blocks, and its miner earns nothing.

Transactions are accepted in that order. A transaction that conflicts with an earlier one is ignored, and the rest of its block still counts. The reward for a blue block is paid by the chain block that merges it, so parallel work is paid for. The miner of a red block earns nothing: that block's reward goes to the miner of the chain block that merges it. This is what makes withholding blocks unprofitable.

## 3.1 Choosing k

The parameter $k$ is the number of parallel blocks the protocol expects from honest miners. With a bound $D$ on network delay and a block rate $\lambda$, the number of blocks made in the window in which a block can be missed is close to a Poisson variable with mean $2D\lambda$. Following the PHANTOM paper, $k$ is the smallest value that such a variable exceeds with probability below a tolerance $\delta$ [2, 8]:

$$k = \min \{ \hat{k} : 1 - \Sigma_{j=0}^{\hat{k}} \, \mathrm{e}^{-2D\lambda} \, (2D\lambda)^j / j! < \delta \}$$

Dahba uses $D$ = 5 seconds, $\lambda$ = 10 blocks a second and $\delta$ = 0.01, which gives $k$ = 124. As in Bitcoin, the security argument assumes that honest miners hold most of the hash power, and here also that blocks cross the network within $D$.

## 3.2 Parameters

| PARAMETER | VALUE |
|---|---|
| Target block interval | 100 ms |
| GHOSTDAG $k$ | 124 |
| Parents per block, at most | 16 |
| Merge set size, at most | 248 blocks |
| Merge depth bound | 36,000 blocks (1 hour) |
| Finality depth | 432,000 blocks (12 hours) |
| Pruning depth | 1,080,000 blocks (30 hours) |
| Coinbase maturity | 1,000 blocks (100 seconds) |
| Difficulty window | 2,641 seconds, sampled every 4 seconds |
| Block mass limits | compute 500,000 · storage 500,000 · transient 1,000,000 |
| Smallest unit | $10^{-8}$ DAH |

These are Kaspa's values for ten blocks a second. All of Kaspa's upgrades are active from the first block, so the chain has no switch-over history.

## 4 Two kinds of funds

**Transparent** funds are ordinary unspent outputs (UTXOs) locked to a public address.

**Shielded** funds are Orchard notes. The chain stores a commitment to each note and, later, its nullifier. It does not learn the owner or the value.

A shielded transaction is an ordinary transaction on a reserved subnetwork whose payload is one Orchard bundle in a fixed layout:

| FIELD | BYTES |
|---|---|
| Header: action count (1), flags (1), value balance (8, signed), anchor (32) | 42 |
| Per action: nullifier, note commitment, value commitment, randomised key, ephemeral key (32 each), note ciphertext (580), outgoing ciphertext (80), spend signature (64) | 884 |
| Binding signature | 64 |
| Proof for $n$ actions | $2{,}720 + 2{,}272n$ |

The payload field already travels through the wire protocol, the database and the RPC, so none of those formats changed. A transaction with two actions is about 9 kB.

The **value balance** is the net amount leaving the pool. When positive it counts as an input of the transaction, when negative as an output. One transaction can therefore do any of four things:

| TRANSACTION | TRANSPARENT SIDE | VALUE BALANCE |
|---|---|---|
| Public to public | inputs and outputs | no bundle |
| Public to private (shield) | inputs | negative |
| Private to private | none | the fee |
| Private to public (unshield) | outputs | positive |

A fully private transaction has no transparent inputs at all. Its fee comes out of the pool.

The bundle's signatures cover the whole transaction except the fields that authorise it, and the transparent signatures cover the payload. Neither half can be lifted onto another transaction.

A wallet derives both kinds of key from one 24-word seed phrase: public addresses by BIP 44 under coin type 0x444148 (written `dahba:…`), and one private address by ZIP 32 (written `dahbaz:…`).

## 5 The shielded pool in the UTXO set

A shielded pool has state: the set of nullifiers, the note commitment tree, and the roots a spend may be proven against. On a chain that state advances one block at a time. On a DAG, blocks made in parallel each extend the state they saw, branches are merged many times a second, and the selected chain can change. The pool needs every operation that coins already have: apply, revert, merge, commit to in the header, prune and sync.

So we store the pool's state as coins. It lives in the UTXO set as entries that no transaction can create or spend. They carry a script version that no output and no miner's payout script may use.

| RECORD | KEY | MEANING |
|---|---|---|
| Nullifier | hash of the nullifier | this note is spent |
| Anchor | hash of a tree root | spends may be proven against this root |
| Tip | hash of its own contents | the tree's frontier and the pool balance. Exactly one exists. |

Four properties follow without new machinery.

- **Parallel blocks and reorganisations.** The records sit in each chain block's UTXO difference. Reverting or merging a branch moves them like any coin.
- **Header commitment.** The records are part of the UTXO commitment that every header carries, so the pool's state is bound into proof of work.
- **Pruning and sync.** A node that syncs from a pruning point downloads the records inside the UTXO set it already verifies against the header, and recovers the tip from that set.
- **Supply audit.** The amount of the tip record is the pool balance. The sum of all UTXO amounts still equals every coin in existence.

The only addition beside the UTXO set is a small store that says which tip record belongs to which block.

## 6 Validation rules

1. **Alone.** The payload is a well-formed bundle of this chain's pool version, and its proof and signatures are valid for this transaction. A node checks this once per transaction. The proofs of one block are verified together.
2. **In its block.** No nullifier appears twice in one block. Like the ordinary double-spend check, this keeps a block's transactions independent, so they can be validated in parallel.

3. **Against the UTXO view.** The anchor has a record, or is the root of the empty tree, and no nullifier has a record.
4. **Through the pool, in order.** The pool balance never goes below zero, whatever the proofs say, and the tree is not full.
5. **Per chain block.** If the block's merge set moved the pool, the tip record is replaced. The new root becomes an anchor if the merge set had a shielded transaction, and otherwise once every ten seconds of blocks: block rewards (section 7.1) move the pool with every chain block, and an anchor for each would only grow the set. A merge set that left the pool alone leaves the records alone.
6. **In the mempool.** A nullifier is tracked like a spent outpoint, so spending a note twice is an ordinary double spend, replacement by fee included.

Rule 3 settles the case a single chain never meets. Two parallel blocks may each spend the same note, and both are valid alone. They are resolved exactly like two blocks that spend the same UTXO: the first in GHOSTDAG order wins, and the other transaction is ignored.

Rule 4 is integer arithmetic on a public number. A flaw in the shielded mathematics could not take out more than was put in.

A spend can use any root a chain block has had. Wallets should pick one a few seconds deep, so a change at the tip does not invalidate the proof.

## 7 Privacy model

What an observer of the chain learns from a shielded transaction:

| HIDDEN | PUBLIC |
|---|---|
| Which notes are spent | That the transaction is shielded |
| Who receives the new notes | The number of actions |
| The value of every note | The value balance: the net amount entering or leaving the pool |
| The link between a note and its later nullifier | The nullifiers, the new commitments and the anchor |
| | Any transparent inputs and outputs, and the fee |

For a transaction between two private addresses the public part is the fee and nothing else about the amount. Shielding and unshielding are different. They show the transparent address and the amount that crossed. Someone who shields an unusual amount and unshields the same amount

soon after has linked the two.

Privacy in Dahba is optional for payments, as in Zcash. A sender hides among the notes in the pool, and the pool holds what people chose to put there and what was mined. CryptoNote's two properties (section 2.4) hold for transactions inside the pool, against the whole pool, but a chain where privacy is a choice will have fewer private transactions than one where it is the rule. In return the public side is auditable: exchange balances, the total supply and the pool's balance (section 5).

### 7.1 Block rewards

On a transparent chain every block names the address that is paid its reward, so a miner's blocks, its balance and what it does with its coins are public. In Dahba a block names a shielded address, and its reward is a note in the pool from the moment it exists. This is a rule and not a choice: a block that names a public address is invalid.

The reward of a block is paid by the chain block that merges it (section 3), which another miner found a fraction of a second later. That miner has no time to make a proof, and needs none. The receiver is the address in the block, the amount is set by the schedule and the fees, and the remaining parts of the note are hashes of the paying transaction and of the output's position in it. Every node derives the same note commitment from these public values and adds it to the tree. Zcash's shielded coinbase [11] also reveals the receiver and the amount, and carries a proof. Ours carries none, because nothing in the note is secret.

What the chain shows of a reward is that a block was found, when, what it paid, and an address. A wallet has $2^{88}$ addresses that nobody else can tell belong together, and a miner names a new one in every block. So the chain shows neither who mined a block nor whether two blocks had the same miner. Spending the note later reveals a nullifier that cannot be linked to it (section 2.3). The supply stays checkable, because the amounts are public and the pool's balance is an entry in the UTXO set (section 5).

The price is that every mined block is a separate note. A transaction can spend eleven notes, which is the number of shielded actions that fit in a block, and the proof costs about a third of a second for each note spent. A wallet sends a larger payment as several transactions and combines notes in the background. Any cheaper way to combine them would show which blocks belong together. For a node, a note commitment costs 0.6 milliseconds, ten times a second.

Three things are not hidden by this. Coins that leave the pool show their amount and time, as in Zcash. A group that mines alone receives every reward, whatever the chain shows. And the network address a block is announced from is a separate matter.

### 7.2 Network addresses

A node's peers see the address it connects from, and see it announce its own blocks and transactions before anyone else does. A node can reach its peers through Tor instead, which is built into it [12]. Such a node only connects out. It opens no port, accepts no connections and tells its peers no address of its own. The mining software starts its node this way.

Tor adds delay. From a home connection we measured 0.17 seconds for a message and its reply through Tor, and 3.6 seconds to join the Tor network from nothing. The parameters of section 3.1 assume up to five seconds of network delay, so a block found behind Tor still arrives in time to be paid. A whole network of nodes behind Tor has not been measured.

The protection is partial. Nodes that hide cannot connect to each other, so the network needs nodes that are open, and those show their address. About a third of Tor's exits carry connections to Dahba's port. Tor is blocked in some places, and an observer of both ends of a connection can match them.

---

## 8 Proof of work

Dahba's mining function is Kaspa's kHeavyHash with its last step replaced.

1. The header, with its nonce, is hashed with cSHAKE256, as in Kaspa.
2. The result, read as 64 four-bit digits, is multiplied by a 64 by 64 matrix derived from the header, as in Kaspa.
3. The product is hashed with BLAKE3 [10], keyed with the string `DahbaHeavyHash`. Kaspa uses a second cSHAKE256 here.

The result is compared with the target. Consequences:

- Machines built to mine Kaspa contain no BLAKE3 stage and cannot mine Dahba. A test in the code confirms that nonces which solve Kaspa's puzzle do not solve Dahba's.
- Graphics cards can mine it, and existing graphics-card miners already implement both hash functions.
- No other coin uses this combination, so there is no existing hash power to rent against the network.
- A node checks a block's proof of work in microseconds. A function that is slow to verify could not be used at ten blocks a second.

The function is not ASIC-resistant, unlike the one CryptoNote [7] argued for. Machines can be built for it, as they were for Kaspa's. Changing it again would be a network upgrade.

Difficulty is retargeted on every block from a window of the last 2,641 seconds, as in Kaspa.

## 9 Supply and incentive

The reward schedule is sized for a supply of 21 billion DAH. Nothing is created at the start. The first block pays nobody, there is no founder or development share in the rules, and every coin comes from a block reward.

The reward schedule has two parts. The launch phase is the first 600 blocks, which is one minute at ten blocks a second. Each of them pays 14,000,000 DAH, 8.4 billion DAH in all, which is 40% of the supply.

After the launch phase a block pays 19.94 DAH, which is 199.41 DAH a second. The reward stays the same for a year and is then cut in half, once a year. In year $y$ after the launch phase, counted from zero, the reward per second is

$$r(y) = 199.41 \cdot 2^{-y} \text{ DAH}$$

70% of the supply is mined in the first year, 85% in the first two, 98% in the first five and 99% in the first six. The reward reaches zero after 35 years, and miners are then paid by fees alone. Bitcoin anticipated the same transition [1].

The schedule counts blocks, not time, and a block's reward depends on its position in the DAG. Blocks found at the same moment just before the reward steps down all receive the higher reward, so the real total can end above the schedule. The schedule itself pays 20,985,989,998.53 DAH, which is 14,010,000 short of 21 billion on purpose. That room covers every yearly step in full, and one block found beside the last block of the launch phase, which adds 14,000,000 DAH. The rules do not cover more than one: the worst case is 248 such blocks, and covering it would remove 16.5% of the supply from the schedule. The number of blocks paid the launch reward is visible on the chain and fixed once the launch phase has passed. With 601 or fewer, the total stays under 21 billion.

Early miners receive a large share of the supply. The chain shows that every coin was mined and what each block paid. It does not show who mined a block, or whether two blocks had the same miner (section 7.1).

Fees are priced on transaction mass, as in Kaspa. A shielded action is charged 40,000 grams of compute mass, which reflects the cost of checking its proof and limits a block to five private transactions. A private transaction costs about forty times the fee of a public one.

## 10 Performance

The figures below were measured on one machine (Apple M2 Max) against a real node producing a block every 100 ms. They are the chain's own delays. A worldwide network adds the time blocks take to travel. Kaspa, with the same consensus, shows transactions in about a second.

| | PUBLIC TRANSACTION | PRIVATE TRANSACTION |
|---|---|---|
| Prepared on the sender's machine | 1 ms | 0.8 s (the proof) |
| Then seen by the receiver | 0.2 s | 0.2 s |
| Room in one block | 245 | 5 |
| Room in one second (ten blocks) | 2,450 | 50 |

The 0.2 seconds is two block intervals: the next block carries the transaction, and the block after it accepts it.

Three measures keep private transactions cheap for nodes. Cheap checks run first: a block that declares more shielded actions than fit is refused before any proof is examined, and an arriving transaction is first checked against the pool, which costs a few lookups. The proofs of a block are verified together: one private transaction alone takes 21.5 ms of one processor core, and the five of a full block together take 30 ms. A proof is verified once: a node remembers the transactions it verified on arrival and does not verify them again in a block.

The present limit of 50 private transactions a second is set by the check on arrival, which handles one transaction at a time. Verifying arrivals in batches as well would allow about 100 a second. Block size would allow 26 per block.

## 11 Pruning and synchronisation

Bitcoin's paper already expected nodes to discard old transactions and keep headers [1]. At ten blocks a second this is a necessity. Nodes keep block bodies for about 30 hours and then drop them. A new node downloads a proof of the chain's accumulated work, then the UTXO set at a pruning point, and checks that set against the commitment in the header.

Because the pool's records are in the UTXO set, a new node receives the nullifiers, the anchors and the tip in the same step and verifies them in the same check. This has been tested with two real nodes, one joining late.

Wallets have a harder problem. To spend a note a wallet must prove a path from the note's commitment to a root, which takes the commitments that came after it. A wallet that starts reading at a block later than the first cannot build that path for older notes. The wallet detects this today and refuses to make the transaction. The remedy is specified but not built (section 13).

---

## 12 Programmability

No virtual machine is added to the base layer. The inherited code can verify zero-knowledge proofs inside a script, both Groth16 and RISC Zero proofs, and can commit to separate lanes of transactions for separate applications. Programs run off the chain. The base layer orders their inputs and checks a proof of the result. An existing execution environment such as the EVM can be run this way as a rollup. The base layer stays fast, and no second execution engine sits inside consensus.

---

## 13 Limitations and open work

Dahba is in development. At the time of writing:

- **No public network.** The genesis block has not been made. It will carry a recent Bitcoin block hash, which proves the chain was not started earlier. There are no seed servers yet.
- **Mining software.** The function is in the node and a processor miner works. A graphics-card miner is not written.
- **Wallets on a pruned chain.** Described in section 11. The fix is an RPC that returns the tree frontier at a block, public funds read from the node's UTXO index, and the compact form of shielded outputs kept beyond the pruning point. This is required before launch.
- **Tree capacity.** The note commitment tree holds $2^{32}$ notes, a size fixed by the proof circuit. Block rewards take ten of its leaves a second and transactions up to forty, which is 2.7 years of completely full blocks. When the tree is full the pool refuses every shielded transaction, unshielding included. A second pool, and a way to move funds into it, has to exist well before then.
- **Anchors do not expire.** At most one anchor record is added per chain block, and none is removed.
- **Pool version.** The code uses the newest pool version the `orchard` library offers. Before launch we will confirm how long that version has been live on Zcash's main network. The fallback is the previous version.
- **Review.** The parts taken from Kaspa and Zcash have been studied and run in production. The code that joins them has its own tests, including a double spend across parallel blocks, a

reorganisation and a late-joining node, but it has not been reviewed by an outside party. That includes the notes of block rewards, which consensus makes from public values without a proof (section 7.1): the one place where a note is not made the way Zcash makes it.

- **Mined funds come in many notes** (section 7.1). A wallet also has to read every leaf of the tree, ten a second from rewards alone, which takes about four minutes for a day of chain. The node should hand over the roots of whole subtrees, so that a wallet can skip the stretches it has nothing in.
- **Privacy is optional** for payments, and a node's network address is hidden only if it uses Tor (section 7.2).
- **Mining hardware** can be specialised (section 8).

---

## 14 Conclusion

We have described a proof-of-work currency that orders blocks with GHOSTDAG at ten a second and carries an Orchard shielded pool. Both parts are existing, public designs, used as their authors wrote them. The one design decision of our own is to hold the pool's state in the UTXO set. The block DAG then treats a private note the way it treats a coin: parallel spends are resolved by the same order, the state is committed in every header, and a new node receives it with the rest of the set. A plain balance check sits under the proofs, so the amount that can leave the pool never exceeds what entered it. A transaction of either kind reaches the receiver in two block intervals on a test network.

---

## References


1. S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. https://bitcoin.org/bitcoin.pdf
2. Y. Sompolinsky, S. Wyborski and A. Zohar, "PHANTOM and GHOSTDAG: A Scalable Generalization of Nakamoto Consensus," Cryptology ePrint Archive, Paper 2018/104. https://eprint.iacr.org/2018/104
3. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," IEEE Symposium on Security and Privacy, 2014. Cryptology ePrint Archive, Paper 2014/349.
4. D.-E. Hopwood, S. Bowe, T. Hornby and N. Wilcox, "Zcash Protocol Specification," version 2026.9.0 [NU7]. https://zips.z.cash/protocol/protocol.pdf
5. D.-E. Hopwood, J. Grigg, S. Bowe, K. Nuttycombe and Y. T. Lai, "ZIP 224: Orchard Shielded Protocol," 2021. https://zips.z.cash/zip-0224
6. S. Bowe, J. Grigg and D. Hopwood, "Recursive Proof Composition without a Trusted Setup," Cryptology ePrint Archive, Paper 2019/1021.
7. N. van Saberhagen, "CryptoNote v 2.0," October 17, 2013.
8. Kaspa developers, "rusty-kaspa," version 2.1.0. https://github.com/kaspanet/rusty-kaspa

9. Zcash developers, "orchard," version 0.16. https://github.com/zcash/orchard
10. J. O'Connor, J.-P. Aumasson, S. Neves and Z. Wilcox-O'Hearn, "BLAKE3: one function, fast everywhere," 2020. https://github.com/BLAKE3-team/BLAKE3-specs
11. J. Grigg, "ZIP 213: Shielded Coinbase," 2019. https://zips.z.cash/zip-0213
12. The Tor Project, "Arti," a Tor implementation in Rust, version 2.8. https://gitlab.torproject.org/tpo/core/arti

---

Dahba is open-source software and an independent project. It is not affiliated with Kaspa, Zcash, Monero or Bitcoin. There is no Dahba coin for sale. Nothing in this paper is financial advice.