TITANFORGE

A proof-of-work layer 1 whose ledger holds value only in shielded notes

Litepaper

Version 1.0, November 2026

titanforge.network

Abstract

TitanForge is a proof-of-work layer 1 whose ledger holds value only in shielded notes. A transparent ledger reveals the parties and the amount of every payment, and a shielded pool beside one leaves the transparent path in use. Every payment is an Orchard transaction with a Halo 2 proof. To keep the supply checkable, mining rewards and their addresses, the balance of each pool and a commitment to the ledger state in every block header are public. The node is a fork of rusty-kaspa. The chain orders blocks with GHOSTDAG at a target of one block per second and is mined with XelisHash-TF. The shielded ledger, the node, a CPU miner, a command-line wallet and an independent supply auditor exist and have run in the project's tests; the encrypted transport, the synchronisation of late joiners, the main network and the stablecoin do not, and none of the code has been audited externally.

Contents

1. Motivation and goals 3
2. System overview 3
3. Shielded payments 4
4. What is public, and why 6
5. Consensus and proof of work 7
6. Monetary policy and supply verification 9
7. Dev fund 11
8. Network and storage 11
9. Wallets 12
10. The planned stablecoin 13
11. Security considerations and limitations 13
12. Current status 14
13. References 14

1. Motivation and goals

1.1 The problem

A public ledger serves two conflicting interests. The payer and the payee want it to say nothing about them; everyone else wants it to show that no coin was created outside the rules. A shielded pool beside a transparent ledger meets the first for those who use it, but the transparent path carries most of the activity, and every crossing between the two reveals an amount and a time. The problem addressed here is to give up neither.

1.2 Design goals and their costs

Shielded only. Value exists only in shielded notes, because a transparent path, however small, is where a privacy system leaks. In return every participant bears the proving cost; exchanges and auditors work with public totals and with viewing keys, which read a wallet's payments and cannot spend; and the usual emergency fallback of paying transparently while a pool is disabled does not exist.

Supply verifiable from public data. Issued and burned supply is committed in every block header and verified by every node, because a shielded-only ledger would otherwise ask its users to trust that the circuit is sound. The cost is that every pool balance, and the recipient address and value of every mining reward, are public.

One block per second on a block DAG. Blocks form a directed acyclic graph (DAG) ordered by GHOSTDAG [1], so blocks mined in parallel stay in the record rather than being orphaned. Parallel blocks carry duplicate transactions, so accepted throughput is below the sum of block capacities.

A proof of work for ordinary processors, as a named variant. The hash is a variant of XelisHash v3 with the project's own keys and input tag, so that mining software built for another coin needs a deliberate change to work on this one. As a result no mining software other than the project's own implements the variant, no GPU miner exists, and whether CPUs and GPUs mine it at comparable rates has not been measured.

1.3 How this paper states facts

Three states are kept distinct. **Built** means the code exists in the repository and has run in the project's own tests. **Specified** means a written rule describes a mechanism the code does not yet implement. **Planned** means work that is intended and not begun, including work the specification requires of people rather than of code: reviews, audits, key ceremonies and release builds. A **working value** may still change before the main network; a **frozen value** is fixed in the specification. The table in section 5.2 marks the state of each consensus parameter of sections 3 to 7.

2. System overview

Node. The node, tfd, is a fork of rusty-kaspa v2.1.0 [18] (ISC licence, copyright © 2022-2024 Kaspa developers), called **upstream** below, and most of the node is that project's work. The transparent transaction format and the set of unspent outputs were removed.

Payments. Payments use the Orchard protocol [4, 5] with Halo 2 proofs [2, 3], through the libraries of the Zcash project, available under either the MIT or the Apache-2.0 licence.

Mining. Mining uses XelisHash-TF, a modified copy of XelisHash v3 [17] (MIT licence, copyright © 2024 XELIS). TitanForge is not affiliated with the Kaspa or XELIS projects.

New rules. The project's own rules are reward notes constructed by consensus rather than by the miner, public pool balances with a per-pool turnstile (section 6.4), a commitment to the supply in every block header, a consensus lockbox for the dev fund (section 7), fixed per-action fees in tiers, and the rules that keep several shielded pools apart within one ledger.

Ordering. GHOSTDAG orders the DAG. Each block is coloured **blue**, well connected, or **red**, poorly connected. **Blue score** counts the blue blocks in a block's past, and **blue work** sums their proof of work. Each block has a **selected parent**, the parent with the most blue work; the **selected chain** is the path of selected parents from the tip, and a **chain block** is a block on it. Each chain block **merges** the blocks it adopts, its **mergeset**. A block's **difficulty window** is the recent stretch of blocks merged along its selected chain from which its difficulty is computed (section 5.3).

Clock. The **DAA score** of a block counts the blocks that contributed to difficulty on its selected chain (DAA: difficulty adjustment algorithm). Every schedule of the protocol is keyed to it, never to timestamps.

Units and networks. One TFN is 100,000,000 rivets, and every amount in consensus is a whole number of rivets. Four networks are defined: a main network, testnet-1, a devnet and a simnet. The node refuses to run the main network: its genesis is a placeholder and its dev-fund address is undefined.

3. Shielded payments

Private here means what is inside a payment: who pays whom, and how much. What the network layer shows today, which node first sent a payment, is the subject of section 8.

3.1 The protocol instance

A **note** is a unit of value with an owner, an amount and randomness; it appears on chain only as a commitment and a ciphertext. A **pool** is one Orchard value pool: one note commitment tree, one nullifier set (section 3.3) and one public balance, holding one asset. Every pool is an instance of the Orchard protocol in its Ironwood version, which is Zcash's own configuration of Orchard: the Action circuit of Zcash's NU6.3 upgrade, which carries the NU6.2 correction of a soundness flaw in its variable-base scalar multiplication [21] and one further public input, and version 3 notes [6]. A proof is a Halo 2 proof, which needs no trusted setup; the verifying key is derived deterministically from the circuit, so a node builds it itself, downloads no parameters and checks the key's fingerprint against a pinned value at start-up.

3.2 Shape

A transfer carries one bundle of exactly 2, 4 or 8 actions, and consensus refuses any other count. An action spends one note and creates one; a payment that needs fewer actions is padded with dummy spends and zero-value outputs, which are indistinguishable on chain from real ones. The action count therefore reveals the size bucket of a payment, not the number of notes it moved.

3.3 Proof, signatures and nullifiers

The wallet makes the proof and the signatures. Together they show that the spent notes exist in the tree at the anchor (section 3.4), that the amounts balance and that the owner approved, without showing which notes were spent. Each action reveals a **nullifier**, a 32-byte value that prevents a second spend of the same note without identifying it. A pool keeps every nullifier forever, and a transaction that repeats one is not accepted. Without the owner's nullifier key, a nullifier cannot be linked to its note. Recipient, amount, memo (a message a payment may carry) and the identity of the spent note do not appear in clear. The recipient's wallet finds its notes by trial decryption under its own keys.

3.4 Anchors

An **anchor** is a root of a pool's tree against which a spend proves that its note exists. The selected chain is divided into buckets of 600 on the DAA score (about 10 minutes), and the anchor of a bucket is the root at its first chain block. Its age is counted from the bucket's nominal position, a multiple of 600, not from that block's own score, and a spend may use an anchor whose nominal age is at least 600 and at most 43,200. A wallet uses the newest valid bucket; one that uses an older bucket shows that it is behind. Because buckets are coarse, an anchor reveals when the wallet last synchronised only to within about 10 minutes, at the cost of latency: a note becomes spendable 600 to 1,200 on the DAA score after the block that appended it (about 10 to 20 minutes), or as little as 420 (about seven minutes) when that block is the first of its bucket, which may lie up to 180 past the nominal position.

3.5 Fees

The fee is a fixed amount per action. Per-block adjustment is impossible, because the fee is fixed inside the proof and blocks are made in parallel; the level is instead a tier, constant for an epoch of 3,600 on the DAA score (about one hour). Consecutive tiers differ by a factor of $2^{1/4}$, and the tier moves at most one step per epoch: up when the previous epoch was at least half full, down when it was at most a quarter full. A transaction paying the current or the previous tier is accepted. The starting tier is 0.0001 TFN (10,000 rivets) per action. Half of the fee is burned and half goes to the miner of the block from which the transaction is accepted.

3.6 Acceptance

A node verifies the proofs and signatures of every block body it processes. Acceptance is separate: the transactions of a chain block's mergeset are applied one at a time in a fixed consensus order, and a transaction is accepted only if it has not expired, its fee is an accepted tier, its anchor is valid, none of its nullifiers is in its pool's set, and the pool balance stays within range. A transaction that fails is skipped and may be accepted later from another block.

4. What is public, and why

4.1 Transaction structure

Every shielded transaction shows its network and rule-set identifier, expiry, pool, action count, anchor and value balance; and for each action its nullifier, note commitment and ciphertext bytes. A transfer's public value balance equals its fee, so the amount paid does not appear. The balance of each pool is public.

4.2 Mining rewards

Mining rewards and the address they are paid to are public on chain, and payments after them are private.

A reward is a **consensus-constructed note**. A block's coinbase, the transaction that lists the rewards of the blocks it merges, names the address its miner wants paid; the chain block that merges it lists the reward in its own coinbase as a descriptor of purpose, asset, recipient and value; and every node derives the same note from the descriptor, the hash of the listing block and the position in the list, and appends its commitment to the current pool's tree. No proof, signature or ciphertext is involved, for two reasons. First, a node checks a coinbase by reconstructing it, and a shielded output whose randomness the miner chose cannot be reconstructed. Second, one Halo 2 proof per block template, the block a miner is given to work on, takes 0.8 to 1.8 seconds on a multi-core processor, which does not fit a one-second block interval. A miner cannot add, remove, redirect or resize a reward note: any deviation disqualifies its block from the selected chain.

For each reward note anyone can see the recipient address, value, asset, pool, purpose, issuing block, leaf position and time; whether and when it is spent, and where the value goes, are not public. This is the same exposure as Zcash's shielded coinbase [7].

A merged blue block inside the difficulty window of the merging block earns its own miner one reward note; outside that window it earns nothing, its subsidy is never issued, and the miner share of its fees is burned. The reward of a merged red block inside the window, and the fee share of a red block outside it, go to the merging block's miner. A merged block lies outside the window when its blue score is more than 2,644 below the merging block's.

Two rewards to one address can be linked to each other and to every other appearance of that address, so a miner should pay rewards to an address kept for mining and change it periodically. In the current wallet that address shares an account with the addresses given to other people, so if the discrete logarithm problem on the Pallas curve were ever broken, the account's incoming viewing key (section 9) could be recovered from the mining address and used to read every payment that account received.

The construction is the project's own. The specification states 10 properties it must satisfy, among them that a spend stays unlinkable to its note although the inputs of the derivation are public, and requires a cryptographer to confirm or correct each in writing before the rules are frozen; that review has not taken place. If the construction were rejected, the fallback would be Zcash's: a shielded coinbase bundle built by the miner, at the cost of one proof per template.

4.3 Pool generations

A pool's commitment tree has 2^{32} leaves. When it reaches 2^{31} the pool starts to seal and a successor opens as the current pool of its asset; after a notice period of 14 days the sealed pool accepts no transfers between users and no new value. Holders move value by a migration, a transaction from the sealed pool to the current one, whose public amount is restricted to 1, 2 or 5 times a power of 10, so that amounts collide instead of fingerprinting a wallet. Each generation starts with only its own notes, so anonymity is within the current pool generation. Only one pool exists on the current networks, and the wallet has no migration flow yet.

5. Consensus and proof of work

5.1 The block DAG

The GHOSTDAG parameter k bounds how many parallel blocks can still be blue. Chain weight is XelisHash-TF blue work alone: no other proof, share, attestation or checkpoint adds to it.

5.2 Block rate, finality and depths

The target is one block per second on every network. A node never reorganises its selected chain below its finality point, 43,200 in blue score (about 12 hours) below its tip. If a heavier chain would require that, the node keeps its chain, raises an alert and waits for its operator. The merge depth bounds how far below a chain block the blocks it merges may lie.

Parameter	Value	State
Block rate	1 per second	Frozen target; measurements in section 5.6
GHOSTDAG k; parents; mergeset	18; at most 10; at most 180	Frozen
Merge depth	3,600	Frozen
Finality depth	43,200 in blue score	Frozen
Pruning depth (section 8.3)	108,000	Frozen
Emission and dev share	sections 6.1 and 7	Frozen
Anchor age	600 to 43,200 on the DAA score	Working value
Fee tiers	starting tier, step, thresholds, epoch and burn share of section 3.5	Working value
Pool sealing	14-day notice; denominations of section 4.3	Working value
Actions per block	72 (fallback 36)	Working value, benchmarks not yet run
Block size	250,000 bytes (fallback 125,000)	Working value, benchmarks not yet run

At the working limits the byte limit binds first: a two-action transfer is about 9,150 bytes, so a full block holds 27 of them (54 actions), or 16 four-action or 8 eight-action transfers. Parallel blocks carrying duplicates keep accepted throughput below these figures, which are arithmetic, not measurements.

5.3 Difficulty adjustment

Difficulty retargets at every block from a window of the blocks that the selected chain has merged, chain blocks and merged blocks alike: every fourth block by DAA index is sampled, 661 samples over 2,644 blocks, about 44 minutes. Difficulty is constant until the window holds 150 samples. Two changes were made to the upstream rule. First, the new target is computed from the mean **work** of the window rather than the mean target, which overshoots when difficulty falls. Second, a **clamp** counts the measured duration of a window as at least two thirds of the expected one, so one retarget can ask for at most 3/2 of the mean work of its window.

The clamp answers the difficulty-raising attack [19]: without it a private chain chooses its own timestamps, replaces the window with cheap blocks and dials one block of almost any difficulty for almost no work. With the clamp no block carries more than 1/440 of the work of its own window, and what remains is a race in which the work of the attacker's blocks grows slowly. In the project's simulation each attempt by an attacker with 45% of the hashrate succeeds at a depth of 3,600 blocks with probability at most 4.2×10^{-8}, and an attacker may repeat attempts at the cost of a new private chain each time. This probability does not fall with depth as the classic race between chains does, and it rises steeply with share, to 1.4×10^{-5} at 46% and 1.5×10^{-3} at 47%, so depth is not relied on above 45%.

The cost is slower settling after a rise in honest hashrate: after a tenfold step the difficulty settles in about 2,400 seconds, against about 760 seconds for the same formula without the clamp and about 2,030 seconds for the upstream rule. Difficulty may fall to the easiest allowed target, but only as fast as the window allows: recovery from a 99% loss of hashrate takes about 7.4 hours, and whether a public network needs a faster easing rule is undecided. The formula and the 2/3 constant are working values until the specification's independent review of the simulation and the acceptance run of section 5.6 are recorded.

5.4 XelisHash-TF

XelisHash-TF is a variant of XelisHash v3: a memory-hard function with a 543,744-byte scratchpad, filled in stage 1 by a stream cipher keyed from BLAKE3 [16] hashes of the input, mixed in stage 3 by two passes of data-dependent arithmetic with one AES round, and digested to 32 bytes by BLAKE3. The specification lists five differences from XelisHash v3:

1. Every stage-1 BLAKE3 call is keyed with a project string.
2. The stage-3 AES round key is the project's own.
3. The 112-byte input ends in a fixed domain tag; the timestamp and nonce offsets are working values.
4. The integer square root uses integers only, so consensus has no floating point; results equal those of XelisHash v3, checked on 14,143,194,111 inputs.
5. A modular power with modulus zero returns 0; XelisHash v3 divides by zero.

Differences 1 and 2 are what stop a XELIS pool from switching to TitanForge; the tag alone does not. The memory, the iteration count and the 32-byte output are unchanged. In one measurement a hash took about 1.4 ms on a thread hashing alone and about 2.8 ms per thread with all 32 threads of the processor hashing.

5.5 Mining software

tf-miner is the reference CPU miner. It mines solo against a node or through tf-stratum, a server of TF-Stratum v0, the project's dialect of the Stratum protocol by which pool miners receive work, on plain TCP. No GPU miner exists; the plan is an open-source, no-fee GPU miner, OpenCL first.

On the devnet and testnet-1 all subsidy goes to XelisHash-TF blocks. The main network's share is undecided: the same, or four fifths with the rest paid through a share path of RandomX, the CPU-oriented proof of work used by Monero, with total emission unchanged and no effect on chain weight. No RandomX code exists.

5.6 Measured block rate

Two 24-hour runs are recorded. The first, of three nodes on two hosts, ran an earlier build of the node, before the shielded ledger existed, on the default depths: 86,657 blocks at 0.9994 blocks per second from the first retarget to the end, within 5% in every whole hour after it, with no red block and no rejected block. The three nodes held the same chain in 8,355 of 8,357 samples; in the other two, one node was 24 and then 43 blocks behind after a relay stall of about 35 seconds, which it had made up by the next sample. The second, of four nodes on three hosts, ran the final code of the shielded ledger with the merge, finality and pruning depths shortened to 300, 600 and 1,500: between 0.976 and 1.020 blocks per second in each of the 23 whole hours after the first, no divergence in 8,498 samples, all 9,671 submitted payments accepted and still accepted on the final chain, and the state recomputed beside the nodes matching every header commitment it covered.

The specification's acceptance scenario asks for 24 hours at one block per second within 5% with no divergence. The first run was too short to reach the pruning depth, and its node had no shielded ledger; the second pruned, on shortened depths. No run yet combines the shielded ledger with the default depths.

6. Monetary policy and supply verification

6.1 Emission

Emission starts at 1.0 TFN per second and halves every 24 months in monthly steps. A month is 2,629,800 seconds, counted on the DAA score. From month 122, about 10.17 years in, a permanent tail pays 0.03 TFN per second: 946,728 TFN a year, about 1.056% of supply in the first tail year and falling.

Nominal supply is about 27.06 million TFN after year 1, 69.28 million after year 4 and 89.65 million at the start of the tail. On a wide DAG the issued supply can differ slightly from the nominal curve.

6.2 No premine

On every network the genesis coinbase has a subsidy field of 0, lists no note and pays nothing; genesis is excluded from every difficulty window, so no later block pays it; and the genesis state holds zero balances. The rule for the main-network genesis, which does not exist yet, is specified: its payload must commit to a public event no older than 24 hours before launch, which shows that the chain was not mined before that event.

6.3 The state commitment

One 32-byte header field, `state_root`, commits to the whole shielded state: each pool's tree, nullifier set, balance and anchors; each asset's issued and burned amounts; the fee state; the dev-fund lockbox (section 7); the fee escrow, the miner shares of fees collected in one chain block and paid by a note constructed in a later one; and a hash chain over the compact blocks, the per-chain-block summaries defined in section 8.3. It is verified at every chain block, and a block with a wrong `state_root` stays a block of the DAG but is disqualified from the selected chain.

After every chain block the supply invariant must hold: for every asset, issued minus burned equals the sum of its pool balances plus what consensus holds in escrow. A transition that would break it has no valid result, and a node that computes one stops with an alert rather than clamping a value or continuing, because clamping a negative tally to zero is how a counterfeit becomes invisible.

6.4 The turnstile and its limit

Value enters a pool only as the public value of a consensus-constructed note or the public amount of a migration, and leaves only through a bundle's public value balance. A bundle that would take a pool's balance below zero is not accepted and is logged as a turnstile event. This is the rule of ZIP 209 [8] applied per pool.

Public totals cannot detect counterfeit notes that stay inside a pool. If the circuit or the pool separation were broken, the turnstile would cap what can leave at what honestly entered; if a sealed pool holds counterfeits, exits stop when its balance reaches zero, holders who have not migrated bear the shortfall, and no rule re-mints, redistributes or refunds.

6.5 Independent audit

tf-auditor is a second, independent implementation of the state rules, written from the specification and the reference models. It recomputes the state of every chain block and reports the first difference. It shares with the node only the transaction parser and digests and two identity strings, so an error in those could escape it. In the project's tests it agrees with the node's transition on 126,551 chain blocks of generated chains.

It reads chain files and generated chains only, so no live chain has been audited with it, and the per-block supply record that the specification requires the node to serve over RPC is not served. Recomputing supply from genesis needs an archival node, because compact blocks do not carry value balances. The auditor is built, and independent audit of a live chain is not yet possible.

7. Dev fund

The dev fund takes 5% of the block subsidy for emission months 0 to 47 and nothing after that. It never takes a share of fees. The total is 3,464,144.86 TFN, 3.871% of the 10-year supply.

Each share accrues to a consensus lockbox, a balance held by consensus and by no key. One note of the whole balance is paid per 3,600 on the DAA score to a consensus constant, the dev-fund address of the network. No key or vote can redirect the lockbox, and changing the address is a hard fork.

Lockbox notes are public, and the fund's spending is read with its full viewing key (section 9), which shows the recipient address, the amount and the memo of each payment the fund makes. For testnet-1 the disclosure record holding that key is in the repository; for the main network the record is specified to be published with the main-network key.

On testnet-1 the address belongs to a single test key that the project holds, and whoever holds its 24 words can spend alone. Main-network custody is specified as a FROST threshold key [13] from a distributed key generation, so that no single person or device ever holds the spending key; participants, threshold and ceremony are not yet chosen.

8. Network and storage

8.1 Transport today

Peer connections use the upstream plaintext gRPC transport, with no TLS and no padded frames, and an admitted transaction is announced to every peer in the next batch. A connected node can therefore see which node first sent a payment, and an observer on the path can read peer traffic. Under the project's rules this transport is acceptable only for test transactions, and a network running it must not be described as private.

8.2 The specified transport and relay

The specification calls for TLS 1.3 [20] on every peer connection, padded frames, and Dandelion++ relay [11], which sends a new transaction along a single path, the stem, before it is broadcast. Once that is built, transport is encrypted and transactions are relayed with Dandelion++; a transaction cannot be tied to an IP address by ordinary peers. That does not hold today, and it will not hold against every observer. An observer of a node's own link will still see that the host runs a node, every peer's address, and the volume and timing of its traffic; without cover traffic it can also tell that a stem frame leaving a node shortly after none arrived was created there. A global observer can follow stem frames hop by hop by timing.

8.3 Storage

A default full node keeps block bodies from the pruning point, the chain block below which bodies are discarded, up to the tip: about 30 to 42 hours. It keeps the nullifier set of every pool forever, and a compact-block archive from genesis: one compact block per chain block, holding what a wallet needs to find its notes and what a future recovery protocol needs to rebuild a pool. The archive is estimated at about 291 GB a year at 27 payments per second. An archival node keeps every body, up to about 7.9 TB a year at full load. Nullifier growth is unsolved: up to about 73 GB a year at the 72-action cap, downloaded in full by every new node.

8.4 Late joiners

The import of a pruning point's shielded state, which a late joiner (a node starting after pruning has begun) needs, is not built. A synchronisation that would move a node to a pruning point it did not validate itself is refused before it starts, so a node offline across a pruning-point move cannot catch up yet. A first-sync risk is unresolved, and the specification treats it as blocking the main network: a fresh node accepts the first well-formed pruning proof it is offered and can be captured by its first syncing peer. A checkpoint rule is specified and not built, and as drafted it does not close this risk.

9. Wallets

tf-wallet is a command-line wallet. A wallet is one 24-word phrase [14], from which keys are derived with ZIP 32 [9]. On the main network the coin type is chosen by the project and not registered in SLIP-0044; on the test networks it is coin type 1, shared with the test networks of other coins. The wallet's file is encrypted under an Argon2id [15] key from a password of at least 12 characters. An account has 2^{88} diversified addresses, and nobody can link two of them without a viewing key of that account. Address strings are Bech32m, jumbled with F4Jumble [10] so that a string matching the first characters of a victim's address cannot be made at no cost. A wallet can export an incoming viewing key, which reveals every payment received and cannot spend; a full viewing key also shows what the wallet sends.

The wallet speaks only to a node on the same computer. The node's wallet-sync service takes no key, address or nullifier as input; the wallet reads whole ranges of blocks and scans them itself. The node therefore learns when the wallet is used, which block ranges it reads and every payment it sends. It does not learn the keys, the wallet's addresses, which notes are the wallet's, which output is change, or the balance.

Only the first account (account 0 of ZIP 32) works. The wallet sends empty memos and reads none, cannot import a viewing key or run view-only, and refuses the main network. Acceptance tests against two running nodes cover mined rewards, payments of 2, 4 and 8 actions, a restore from the words, a reorganisation, and restarts across pruning-point moves on shortened depths; they have not run on the default depths.

Desktop and web wallets, a wallet daemon for exchanges and pools, and payment proofs, which show one payment without a viewing key, have no code yet. A mobile wallet is planned, not before the main network, and light synchronisation for phones is a research question.

10. The planned stablecoin

Tungsten (TGN) and its reserve share Girder (GIRD) are planned, and they activate only when criteria are met. Nothing in this area is active: the node holds only the names, tickers and reserved fields, activation is set to never on every network, and conversion transactions are refused. A draft specification and an integer reference model exist, and most of their numbers are working values.

The design is a hardened Minimal Djed [12]. A reserve of TFN backs tungsten, which targets 1 USD, and girder holders own the equity of the reserve and absorb its risk. The system mints tungsten only while the reserve ratio is at least 500%, and girder only up to 1,000%; both bounds are frozen in the specification. A conversion hands its input to consensus, which holds it in escrow and settles three epochs later at a price that did not exist at request time. Each operation settles at whichever of three prices is least favourable to the user: the epoch price, a 24-hour median and the unclamped oracle median. Redemption of tungsten has no ratio gate and, in normal operation, no cap, but it is priced at the least favourable of the three prices and so is not at par; while fewer than five oracle slots are fresh there are no conversions at all. Nine publicly identified oracle operators would sign prices; none has been chosen.

Activation takes a hard-fork release, and no on-chain vote or key can trigger it. The criteria include the security cost, oracle performance on a test network, two independent external audits, legal opinions and a main network at least 180 days old.

The design is not formally verified, because Djed's proofs cover the real-number protocol and not this integer variant. A fall in the price of TFN by more than 1 minus 1/r, where r is the reserve ratio when the fall begins, breaks full backing: from exactly 500% an 80% fall takes the ratio to 100%, and beyond that redemption is pro rata and the shortfall is not made good. The 500% figure is a minting gate, not a floor: the ratio may already be lower when a fall begins. Price formation rests on five of the nine operators. The holders of girder cannot leave below 500%, and the kinds, amounts, both addresses and timing of conversions are public, so privacy is weaker here than in payments. Legal classification is unresolved.

11. Security considerations and limitations

Audits. Nothing is audited. External audits are a precondition of the main network: the rule that keeps the node from running one is removed only at a readiness review whose requirements include them, the encrypted transport, checkpoints, the emergency-freeze template, the real genesis, a published dev-fund viewing key and the stablecoin running on a test network.

New constructions. The consensus-constructed notes have not had the cryptographer's review the specification requires (section 4.2). The multi-pool ledger rules are new; the formal results inherited from Zcash cover one pool and its balance only.

Internal checks. The project's own checks, which are not audits, are more than 2,100 automated tests, 26 fuzz harnesses of the project's own code, about 20 reference models in Python with committed vectors, and the second implementation of the state rules (section 6.5).

Counterfeits. Public totals bound what leaves a pool, not what is inside it (section 6.4).

Quantum adversaries. Notes are quantum-recoverable. Under ZIP 2005 [6] a note's commitment randomness is derived by hashing the note's fields rather than chosen freely, so that a future recovery protocol could let an owner prove ownership of a note and move it after the discrete logarithm problem is broken, and every node keeps the commitments and nullifiers of every pool, in their order, for that protocol. Consensus derives its own reward notes this way; it cannot check that a user's note was so derived, and no recovery protocol exists. That is a recovery property, not security against a quantum adversary. An adversary who breaks discrete logarithms could break balance and could read recorded ciphertexts.

Emergency template. The specification includes a pool freeze (a soft fork) and a verifying-key swap (a hard fork); neither has been drilled. While a pool is frozen there is no payment path in it.

Network and synchronisation. The transport is plaintext today (section 8.1); late joiners cannot synchronise yet, and the first-sync capture risk is unresolved (section 8.4).

Operation. A node that finds a fault in its own data or code halts: on a broken supply invariant, on a verifying key that is not the pinned one, or on stores that contradict what it wrote. Every halt condition is a check of the node's own data or code; no message from a peer or an RPC caller is among them.

12. Current status

State	Components
Built	The node tfd with the shielded ledger, transaction pool, templates, state commitment and halt behaviour; tf-miner; tf-stratum; tf-wallet; tf-auditor; tf-keytool, the offline key command for test networks; the testnet-1 parameters and dev-fund record
Specified, not built	Encrypted transport and Dandelion++; synchronisation of the shielded state for late joiners; checkpoints and signed releases; the 14-day detail window for memos; the per-block supply record over RPC; the main-network genesis; the freeze and key-swap drills; the stablecoin and its oracle
Planned	GPU miner; pool software; desktop, web and mobile wallets; a wallet daemon; payment proofs; an explorer; external audits; the cryptographer's review of the reward construction; the main-network dev-fund key; reproducible builds

The following do not exist: a TFN coin, a main network and a public test network. No binary has been released. Testnet-1 has not been launched. The source code has not been published at the time of writing, so a reader cannot yet check this paper against it.

13. References

1. Y. Sompolinsky, S. Wyborski and A. Zohar. PHANTOM and GHOSTDAG: A Scalable Generalization of Nakamoto Consensus. IACR Cryptology ePrint Archive, report 2018/104, 2021. https://eprint.iacr.org/2018/104
2. S. Bowe, J. Grigg and D.-E. Hopwood. Halo: Recursive Proof Composition without a Trusted Setup. IACR Cryptology ePrint Archive, report 2019/1021, 2019. https://eprint.iacr.org/2019/1021
3. The halo2 Book (zcash/halo2). Accessed 2026. https://zcash.github.io/halo2/
4. D.-E. Hopwood, S. Bowe, T. Hornby and N. Wilcox. Zcash Protocol Specification, version v2026.7.0-399-gff594d [NU6.3]. 2026. 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. D.-E. Hopwood and J. Grigg. ZIP 2005: Ironwood Quantum Recoverability. 2025. https://zips.z.cash/zip-2005
7. J. Grigg. ZIP 213: Shielded Coinbase. 2019. https://zips.z.cash/zip-0213
8. S. Bowe and D.-E. Hopwood. ZIP 209: Prohibit Out-of-Range Chain Value Pool Balances. 2019. https://zips.z.cash/zip-0209
9. J. Grigg and D.-E. Hopwood. ZIP 32: Shielded Hierarchical Deterministic Wallets. 2018. https://zips.z.cash/zip-0032
10. D.-E. Hopwood, N. Wilcox, J. Grigg, S. Bowe, K. Nuttycombe, G. Pfeil and Y. T. Lai. ZIP 316: Unified Addresses and Unified Viewing Keys. 2021. https://zips.z.cash/zip-0316
11. G. Fanti, S. Bojja Venkatakrishnan, S. Bakshi, B. Denby, S. Bhargava, A. Miller and P. Viswanath. Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2(2), 2018. https://arxiv.org/abs/1805.11060
12. J. Zahnentferner, D. Kaidalov, J.-F. Etienne and J. Díaz. Djed: A Formally Verified Crypto-Backed Pegged Algorithmic Stablecoin. IACR Cryptology ePrint Archive, report 2021/1069, 2021. https://eprint.iacr.org/2021/1069
13. C. Komlo and I. Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. Selected Areas in Cryptography (SAC), 2020; IACR Cryptology ePrint Archive, report 2020/852. https://eprint.iacr.org/2020/852
14. M. Palatinus, P. Rusnak, A. Voisine and S. Bowe. BIP 39: Mnemonic code for generating deterministic keys. 2013. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki
15. A. Biryukov, D. Dinu, D. Khovratovich and S. Josefsson. RFC 9106: Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. IETF, 2021. https://www.rfc-editor.org/rfc/rfc9106
16. 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
17. XELIS. xelis-hash, source code, commit f48262b. MIT licence, 2026. https://github.com/xelis-project/xelis-hash
18. Kaspa developers. rusty-kaspa: Kaspa full-node reference implementation and related libraries in the Rust programming language, v2.1.0. ISC licence, 2026. https://github.com/kaspanet/rusty-kaspa
19. L. Bahack. Theoretical Bitcoin Attacks with less than Half of the Computational Power (draft). arXiv:1312.7013, 2013. https://arxiv.org/abs/1312.7013
20. E. Rescorla. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3. IETF, 2018. https://www.rfc-editor.org/rfc/rfc8446
21. D.-E. Hopwood. ZIP 257: Deployment of the Orchard Temporary Vulnerability Mitigation and NU6.2 Network Upgrade. 2026. https://zips.z.cash/zip-0257

Notice

This paper describes a project in development. Its rules and values may change. No TFN coin exists, the coins of a test network have no value, and nothing here is financial advice or an offer of anything.