

TECHNICAL SUMMARY

ZKrise: what the upgrade fixes, and how

ZKrise is one ZKas network upgrade. It makes proof of work cover everything a node used to take on trust: the shielded anchors below a fast-sync pruning point, the developer-fee balance, and which ZKas block a merged-mining solution belongs to. It also accrues miner rewards in consensus state and ends the developer fee. Every consensus change and the dev-fee end activate together at one score: **mainnet DAA 6,983,500** (about 15 Oct 2026, 15:00 UTC). Upgrade every node, pool, miner and walletd before **DAA 6,897,100** (about 14 Oct, 13:40 UTC), where the coinbase merge-mining tag ban begins and an un-upgraded node splits from the network. Below the fork the new binary produces and accepts exactly what the running network does.

The causes

Each weakness below comes from something consensus did not commit, so a node had to trust a peer, a witness or a convention. The fix in every case is to make the missing value part of what proof of work covers.

Weakness	What was not committed	ZKrise
A fast-syncing node takes the anchors below its pruning point from its sync peer (ZK-01). A lying peer can plant anchors, shift their age, cut the window short, oversize it, or rely on data honest servers no longer hold.	The anchors of the last max-anchor-age blocks below the pruning point.	An anchor window committed in every chain block's state root (section 2).
A spend carries only a tree root, which says nothing about which side of a chain split it came from. With miner accrual (section 4) many consecutive chain blocks add no note and share one root.	Which block the anchor belongs to.	The spend names its anchor block, covered by the signature hash (section 1).
One kHeavyHash solution can be the proof of work of two ZKas blocks: a ZKas block is itself shaped like a valid merged-mining parent.	A one-to-one link between a parent solution and a ZKas block.	Bound commitment; ZKas nonce derived from the parent; ZKas coinbase may not carry a commitment (section 3).
Different witnesses for one header give different proof-of-work levels, so nodes can disagree on parents-by-level (ZK-02).	The witness (it is outside the block hash).	The nonce fixes the parent, so the hash fixes the witness (section 3).
A merged-mining commitment names a block hash only.	The chain.	Genesis hash inside the commitment (section 3).
The developer-fee balance carried between payouts is taken from the sync peer.	dev_accrued.	Committed in the state root (section 2); refused on import where consensus fixes it to zero.
Every rewarded block mints its own note, mostly to the same script.	-	One accrual slot in the state root, paid every 100 DAA or on a payout-script change (section 4).

1. Spends name their anchor block

- A bundle sets flag bit 3 and carries a 32-byte block hash after the anchor. The field is covered by the transaction's signature hash.

- Consensus resolves the anchor from the named block only: the anchor is valid if that block's own committed tree root equals it or, for a block below this node's pruning point, if the pair (anchor, block) is among the import-verified window pairs. The block must then be a selected-chain ancestor that is at least 600 blue score deep (anchor depth) and at most 27,000 (maximum anchor age) below the including block. No search over producers: orphans, identical sibling roots and peer rows for other blocks cannot influence the verdict.
- Format gating: the DAA score of the carrying block decides. Before the score the field is refused, from it is required. Context-free checks accept either form; the exact form is enforced in header context, so an old and a new node judge every pre-activation body identically. Fuzzed (100,000 mutations): an accepted encoding re-encodes to itself; with bit 3 set the two formats never both accept the same bytes; without it they always agree.
- Side effect: about ten minutes after a chain split, transactions no longer replay across it, because the named block exists on one side only.

2. The anchor window and state root v1

Item	Definition
Entry (one per chain block)	<code>block[32] selected_parent[32] tree_root[32] blue_score u64</code>
Bucket	<code>blue_score / 1,000</code>
Bucket chain	<code>H("zkas_anchor_win1"; prev entry)</code> , folded per bucket, starting from 32 zero bytes
Retained	The newest bucket and <code>ceil(max_age / 1,000) + 1 = 28</code> below it (29 at most) on mainnet
Order	Strictly increasing blue score; a list that repeats or reverses a score is refused
Commitment	<code>H("zkas_anchor_cmt1"; count u32 (index u64 chain)...) </code>
State root v1	<code>H("zkas_state_root1"; root0 dev_accrued u64 window slot)</code>
Slot encoding	<code>script_version u16 script_len u32 script amount u64</code> (length-prefixed)
Hash	BLAKE2b-256, 16-byte personalisation

`root0` is the unchanged pre-fork root (tree, nullifier accumulator, supply). A different personalisation means no v0 root can equal a v1 root. Each chain block's coinbase commits its selected parent's root, as before, so the window is bound to proof of work through the existing commitment. A block's window depends only on its selected parent's window and itself, so reorgs need no special handling. Entries are parent-linked, so a node without a chain index below its pruning point can still enumerate them.

Fast-sync import (receiver)

- 1 The pruning-point metadata is decoded and bound before the nullifier stream is consumed; every count is capped. Window entries are capped at $(keep + 2) \times 1,000$ before any hashing.
- 2 Pre-fork pruning point: the v0 binding as today, plus two rules consensus fixes regardless of any root: no miner slot, and no dev balance before accrual starts (DAA 757,000) or from the dev-fee end.
- 3 Post-fork pruning point: re-fold the entries (strict order), require the last entry to be the pruning point, recompute v1 from the transferred root0, dev balance, window and slot, and require it to equal the root committed by the pruning point's child. Changing, dropping, duplicating, reordering, truncating or appending any entry fails here.
- 4 Anchor pairs are kept only when their source block is stored, is a chain ancestor and lies inside the window (header blue score). Peer pairs for anything the window covers are ignored.

- 5 The old state is cleared and the new one written in one batch: there is no moment with an empty nullifier set, and a failed verification leaves the old state untouched.

3. Merged mining: one solution, one block

```
commitment = H_block("zkas_aux_bind1" || genesis || H_header(ZKas header with nonce = 0))
parent tag  = "ZKM1" || hex(commitment)           (64 lowercase hex, exactly once)
ZKas nonce  = first 8 bytes, little-endian, of hash(parent header) (recomputed from fields)
parent      = must not also carry exactly one well-formed "ZKMM" commitment
ZKas coinbase may carry neither a well-formed ZKMM nor ZKM1 commitment
```

- The full block hash covers the nonce and the nonce is the parent's hash, so a block hash fixes its parent: one block, one witness, one level. The level stays the real work of the parent (not a lower bound), and is the same on every node because the witness cannot be swapped.
- A ZKas block cannot be a parent: its coinbase may not commit. A parent valid under both rules is impossible (fuzzed over random parents with 0-3 tags of each kind: no parent is ever accepted by both).
- The rule is chosen by the header's own DAA score everywhere proof of work is evaluated: header processing, pruning-proof validation and level re-derivation, trusted headers.
- Pool order: build the ZKas header, commit with the nonce zeroed, mine the parent, then set the ZKas nonce from the solved parent. One parent template gives a different ZKas block per solving nonce; pools submit the first.
- Live census before choosing the score: 120,106 mainnet blocks, none carries a commitment in its own coinbase.

4. Miner accrual and the coinbase bound

- Payments are walked in coinbase order (mergeset blues, then the red reward). A payment to the slot's script adds to it; a different script pays the slot as one note and starts a new slot. The slot is flushed when the DAA score crosses a multiple of `miner_accrual_payout_interval` (100 at 1 BPS, about 100 seconds).
- Conservation, per block: outputs + new slot = parent slot + payments. Checked against an independent model on 300 random chains of 120 blocks (36,000 coinbases) with random mergesets, reds, non-DAA blocks, fees, payout scripts and every combination of activations.
- Coinbase output bound $k+4$ (22 at $k = 18$), the exact worst case: up to $k+2$ payments, each opening a new streak, plus the interval flush, plus a dev note.
- The coinbase-rewards RPC reports the sink's slot (`pendingRecipient`, `pendingValue`), so pools do not read accrual as a missing reward.

5. Developer fee end

```
dev_fee_end = security_fork_activation = mainnet DAA 6,983,500 (about 15 Oct 2026, 15:00 UTC)
block DAA < end : 5% of subsidy accrues; paid on each 1,000-DAA crossing (unchanged)
block DAA >= end : dev cut = 0; the miner keeps the whole subsidy
first block with DAA >= end (its selected parent below): pays the carried balance as one note
afterwards          : no dev output
```

- Judged by the DAA score of the block whose coinbase pays. A DAG block can step over the score; "at or past" handles it, and every node draws the line at the same block. Nothing is burned or stranded; issuance is unchanged.

- The dev-fee end is unified with the security fork in this release: both take effect at DAA 6,983,500, so an operator upgrades once. An earlier design set it at the second halving (DAA 15,778,800); it was brought forward.
- After the end the honest carried balance is zero; a pruning point claiming otherwise is refused on import, before or after the fork.

6. Activation

Network	security_fork_activation	dev_fee_end_activation	Overrides
mainnet	DAA 6,983,500	DAA 6,983,500	Refused by node, walletd, bridge and miner
testnet	never()	never()	Override-params file
simnet / devnet	always() / 1,500	never()	Override-params file; walletd, bridge and miner by env/flag

How this differs from the earlier anchor-commitment proposal

An earlier proposal replaced node-side verification of a peer's anchor list with an append-only anchor log committed in the state root. ZKrise keeps its central idea, that proof of work should secure the anchors and a syncing node should check them with one comparison, and differs in the following ways:

Aspect	Earlier proposal	ZKrise	Consequence
What a spend commits to	The tree root only; the node maps root to block through the log	Root and the named block, signature-covered	Under today's rules every chain block mints a coinbase note, so roots are unique and a log can map root to block. ZKrise's accrual makes consecutive blocks share a root, so the block must be named; age is then measured from the exact block, and replay across a split stops. Cost: wallets must update and each spend is 32 bytes larger.
Committed structure	Merkle mountain range over every anchor since activation; leaf (anchor, block, blue score)	Sliding window of 29 buckets of folded entries (block, parent, root, blue score)	Both are one comparison against the committed root. The window's state is bounded and old buckets drop out; the entries are parent-linked so the window can be walked without a chain index.
Fast-sync payload (mainnet)	Log prefix plus leaves, about 0.9 MB	Window entries, up to 30,000 x 104 bytes (about 2-3 MB)	The earlier proposal sends less.
Other values in the root	Anchor log	Anchor window, dev_accrued, miner slot	The dev balance and the accrual slot can no longer be misstated by a sync peer.
Merged-mining fork	Level taken from the block's own target (witness-independent lower bound); witness size limit	Genesis-bound commitment, nonce derived from the parent, coinbase commitment ban, dual-tag exclusion; size cap without a fork	Both remove level disagreement. Only the bound form stops one solution from validating two blocks and binds the commitment to the chain; the level remains the parent's real work.
Activations	Two switches (log, AuxPoW)	One switch for all consensus changes, including the dev-fee end	Operators upgrade once.
Transition	About one pruning depth plus one anchor age	The same (about 37.5 hours on mainnet)	Anchors older than the first committed entry still come from the peer until then.

The earlier package's five fast-sync symptoms, and where each is closed in ZKrise:

Symptom	Closed by
Peer plants anchors that never existed	Window entries must re-fold to the PoW-committed root; spends must name a block that produced the root
Peer assigns a real anchor to the wrong block, shifting its age	Entries carry block and blue score inside the commitment; the spend names the block
Peer cuts the window short or mislabels its bottom	The window must end at the pruning point and reproduce the committed bucket set; anything missing changes the commitment
Peer sends an oversized window	Entry count capped at $(\text{keep} + 2) \times 1,000$ before any hashing
Honest servers lose the data a window must start from	Window rows are kept for $\max(2 \times \text{max age}, (\text{keep} + 1) \times 1,000) = 54,000$ blue score below the pruning point, more than the 29,000 the export needs

Does one commitment fix every problem?

For the anchors, yes: committing them so that proof of work covers them replaces the earlier node-side defenses (replay, per-block bindings, coverage checks, size limits, checkpoint retention) and the bugs later reviews kept finding in them, with one comparison that every node also checks on every block. That is the reason ZKrise commits its anchor window. It does not close everything a fast-syncing node took on trust, nor the merged-mining problems, and some needs remain in either design:

Problem	An anchor commitment alone	ZKrise
Planted anchors; wrong block or shifted age; window cut short or mislabeled	Closed	Closed the same way
Oversized window	Still needs a count cap before hashing	Capped at $(\text{keep} + 2) \times 1,000$ entries
Servers losing the data a window starts from	Reduced: a missing record fails the sync instead of fooling it, but servers must still keep the records	Same; records kept 54,000 blue score below the pruning point
Developer-fee balance at the pruning point	Open: not in the log	Committed in the state root
One merged-mining solution counted as two blocks	Open (the accompanying AuxPoW fork fixes level and size only)	Closed by the bound commitment
A spend valid on both sides of a split	Open	Closed by the named anchor block
First ~37.5 hours after activation	Older anchors still from the peer	Same

Trade-offs

- **Activation-dependent.** The consensus fixes apply from the security-fork score, not on upgrade. Until then mainnet runs the old rules, including a coinbase output cap $(k+2)$ below the honest worst case; choosing an early score matters. The no-fork hardening below applies on upgrade.
- **Consensus-critical.** A disagreement about the window, the slot or the bound commitment is a chain split. Tested against exactly that: old and new nodes side by side across the score, deep reorgs across payouts and the boundary, and fresh nodes importing a post-fork pruning point.
- **Wallet change.** Spends carry the anchor block, so every wallet must update before the score. An old wallet's spend is refused, never misapplied.

- **Unified activation.** The dev-fee end and the security fork both activate at mainnet DAA 6,983,500 (about 15 Oct 2026, 15:00 UTC). Upgrade before DAA 6,897,100 (about 14 Oct) or the node splits from the network.

Also in the release, without activation

- Invalid-proof relay disconnects the peer; failed verifications are never cached.
- Pruning-proof GHOSTDAG data is checked against header work; malformed mergesets and wrong trusted-header versions are refused. Aux witnesses are capped at 256 KiB and a nesting depth of 4.
- History backfill that is not fully verified is purged, never marked complete. The pruning-point nullifier export is verified against the committed MuHash before it is cached.
- Startup self-check of the live nullifier set against the committed snapshot at the sink (holding the virtual upgradable lock, so a block resolved during the walk cannot raise a false alarm); on mismatch, templates are refused.
- Block-template cache never patches a coinbase carrying a red reward for another miner; it rebuilds.
- RPC and peer-input panics became errors; coinbase-rewards RPC capped at 256 recipients.
- Zakura 1.0.1 to 2.2.0: circuit, parameters and vectors byte-identical; 1,513 mainnet proofs and 2 x 40 fresh proofs verify identically under both versions in both directions, 0 tampered proofs accepted; the node refuses to start if the verifying-key fingerprint differs from 1bee049a . . . 6591fb.
- Wallets: coinbase commitments derived, not trusted; a repeated nullifier halts the database instead of forking it; seed-overwrite guard; multiparty signatures checked on arrival; change must decrypt with the wallet's own key.

Test evidence

Check	Result
Unit and integration suites on the release tree	consensus 162, shielded-core (circuit) 161, consensus-core 77, mining 64, rpc 131/7/8, walletd 25, p2p 8, pow 17, miner 4, signer 5, gateway 6, WASM signer 13, pool bridge 82 + 122: all passed
Coinbase differential test	36,000 coinbases across 300 random chains match an independent model; value conserved per chain
Fuzzing of peer-facing decoders	Bundle decoder 100,000 mutations; pruning-point metadata 20,000 corruptions; merged-mining rules 400 random parents: no panic, canonical re-encoding, no parent accepted by both rules
Window tampering property	200 random windows; field change, drop, duplicate, reorder, truncate, append, repeated score: every one refused or detected
Deliberately broken builds	Seven in the final round (final dev payout removed, interval rule changed, cut kept after the end, dual-tag rule removed, template patching restored, import guard disabled, final payout removed end to end): all caught
Mainnet, fork off	Full sync with the new binary: 123,142 blocks, 0 rejected, shielded tree identical to the live node (12,711,966 leaves); census of 120,106 coinbases: none carries a commitment
Live devnet, old binary as a peer	Agreed to the fork score, stopped at the first post-fork block and stayed up
Live devnet, new nodes	Native and merged miners across the fork, identical tip and tree; legacy merged form rejected after the score; deep reorg (360 vs 175 blocks) healed; post-fork payments both ways credited exactly

Check	Result
Fresh node from a post-fork pruning point	Window, slot and dev balance imported and bound; identical tree; a second wallet daemon showed identical balances
Double-credit probe	Fork off: one solution yields two blocks. Fork on: legacy form neither, bound form exactly one
Security fork and dev-fee end together	One final dev note at the first block past the end, none in the 1,228 chain blocks after; both nodes identical; 0 rule violations
Independent adversarial reviews	Four reviewers on the full diff, each finding re-verified in code; confirmed defects fixed with a regression test each

Not caused by this release, stated for completeness: 18 of 33 inherited upstream integration tests assume Kaspas's transparent coinbase and fail identically on the pre-release base (c4fc928). A crash after partition, heal and prune requires a pruning depth below finality plus the reachability reindex slack, a test-only configuration; the current mainnet binary shows it too, and mainnet's depths (108,000 vs 43,200 + 16,384) exclude it.

Files

Area	Files
Anchor block, bundle format	shielded-core/src/bundle.rs, consensus/src/processes/transaction_validator/*, virtual_processor/processor.rs (resolve_shielded_anchor)
Window and state root	shielded-core/src/{anchor_window,commitment}.rs, consensus/src/processes/shielded.rs, model/stores/shielded.rs, database registry 95-97
Merged mining	consensus/core/src/auxpow.rs, consensus/pow/src/*, header processor, pruning proof
Accrual, dev-fee end	consensus/src/processes/coinbase.rs, virtual_processor/{processor,utxo_validation}.rs, consensus/core/src/config/params.rs
Clients	zkas-walletd, miner (build_aux_pow_bound), pool bridge (merged.rs, kaspapi.rs, share_handler.rs), gateway