Kaos: A Post-Quantum Electronic Cash

Radium Labs LLC
https://radiumlabs.net
https://www.kaos.cash

October 26, 2026

Abstract

This paper describes Kaos, an electronic cash system in which every layer is post-quantum. Transactions are signed with ML-DSA or Falcon, and new schemes can be added or deprecated through routine upgrades. Peers connect over ML-KEM sessions with no fallback to pre-quantum cryptography, and validators prove their identity with post-quantum keys. Consensus runs on a code-exact C++ port of Avalanche's Snowman, which never collects or aggregates signatures and so scales with large post-quantum keys. A producer rota replaces Snowman++'s proposer windows, and the beacon system records node uptime on chain and distributes rewards, with a fixed delegate commission and a permanent tail emission. vTPS lets Kaos's throughput be compared directly with that of pre-quantum chains. In the wallet, Shamir's Secret Sharing replaces seed phrases, so no single stolen secret exposes a user's funds.

1 Introduction

Nearly every cryptocurrency in use today protects its users' funds with elliptic-curve signatures. A sufficiently capable quantum computer running Shor's algorithm would break that cryptography. An attacker could derive private keys from public ones, forge transactions, and take control of funds. Swapping out the signature algorithm alone does not solve the problem. Signatures are one layer among many, and a system is only as quantum-resistant as its weakest layer.

Kaos is an electronic cash system built from the ground up to be fully post-quantum. This paper describes its three components: the Kaos engine, the Kaos protocol, and the Kaos application.

1.1 What "fully post-quantum" means

We define a cryptocurrency as fully post-quantum when every layer that protects its data, user assets, and network rules remains secure against an adversary with a large-scale quantum computer. A fully post-quantum system:

1. **Supports multiple signature schemes.** It deploys more than one post-quantum signature scheme, and it can add more through a routine network upgrade.
2. **Secures every connection.** Network connections use post-quantum algorithms, with no fallback to pre-quantum ones.
3. **Uses post-quantum validator identities.** Validators are identified and authenticated by post-quantum keys, not by certificates built on pre-quantum cryptography.
4. **Uses post-quantum transaction signatures.**

5. **Scales consensus to post-quantum signature sizes.** Consensus handles the larger signatures that post-quantum schemes produce, without relying on algebraic aggregation.
6. **Uses hash-based addresses.** Addresses are built on hash functions with at least 256 bits of output, not on public-key algebra that a quantum computer could invert.
7. **Protects long-term encryption.** Data encrypted at rest uses 256-bit symmetric encryption, and any public-key encryption uses post-quantum key encapsulation (KEM).
8. **Can retire compromised keys.** If a vulnerability is found in a post-quantum scheme, the affected keys can be deprecated.
9. **Offers purely post-quantum clients.** Client software can interact with the network using only post-quantum mechanisms.
10. **Does not weaken its schemes for legacy compatibility.** It does not modify post-quantum signature schemes to support pre-quantum seed phrases.

Kaos meets all ten. The sections that follow describe how.

2 Kaos Engine

2.1 Signature Schemes

Out of the box the following signature schemes are supported [13, 14]:

- ML-DSA-44
- ML-DSA-65
- ML-DSA-87
- Falcon-512
- Falcon-1024

The signature schemes can be expanded to add new schemes with a simple network upgrade rule. Just define the time when it should go active and integrate it into the node and client codebases.

Schemes are deprecated the opposite way. A signature scheme is never removed from the node, because transactions signed with it must remain verifiable by any node syncing the chain from the beginning. Deprecation happens in clients instead: a deprecated scheme is removed from the list of key types a client can create, so no new keys of that type are made, and users holding such keys are told through published notices to move their funds to a supported type.

2.1.1 Size Comparisons (Public Key + Signature)

This chart shows the length of combined public key and signature size in bytes. Note that some signature schemes use variable-length signatures; the maximum size is used for each signature size below:

Type	Size (Bytes)
ML-DSA-44	3732
ML-DSA-65	5261
ML-DSA-87	7219
Falcon-512	1649
Falcon-1024	3255

2.2 Containers

Kaos does not use blocks. A block is, by name and by convention, a block of transactions, and the Kaos engine is not tied to transactions. Instead, Kaos orders generic containers, each carrying a single container object of whatever type its chain defines.

The engine can run any number of chains in parallel. A chain is defined by three things:

- a chain ID,
- a container object type, and
- the rules for handling that object.

Every container header records its chain ID, height, parent ID, timestamp and size, so each chain is an ordered, hash-linked ledger finalized by consensus. What differs from a blockchain is that the ledger's contents are defined by the chain, not by the engine.

The main chain, Aurora, subclasses the container object as a transaction container, `container_txn`, which carries only transaction objects. Aurora produces a container only when:

1. there is at least one outstanding transaction in the transaction pool, or
2. a beacon transaction is due to be included in the ledger.

2.3 Consensus

Kaos reaches agreement on containers with Snowman, the consensus protocol behind the Avalanche network. We chose it because it finalizes quickly across a global network and never needs to collect or aggregate signatures, which makes it a natural fit for large post-quantum keys and signatures. Snowman is a demanding protocol to implement correctly, and Kaos does not approximate it. Kaos includes a code-exact C++ port of the Snowman consensus engine from AvalancheGo [11], together with the complete test suites of AvalancheGo's `snow/consensus/snowball` and `snow/consensus/snowman` packages, hundreds of tests in all, every one of which passes. The port matches AvalancheGo v1.14.0 (Granite). Kaos replaces Snowman++'s proposer-window layer with its own producer rota, described below, to address two weaknesses we identified in it.

How it works. Snowman is a metastable consensus protocol from the Avalanche family [1]. A producer pushes a new container to k nodes. A node with no preference adopts the first container it hears about, fetches it, and begins polling k randomly sampled nodes for their preferred container. A poll succeeds when at least α of the k nodes report the same preference. A node finalizes a container after β consecutive successful polls. Consensus and container gossip share one message flow.

Why it suits post-quantum signatures. Snowman never collects quorum certificates, so it needs no signature aggregation.

Idle behavior. Kaos produces a container only when the transaction pool is non-empty or a beacon is due. Snowman does not require blocks at fixed intervals, so the chain idles when there is no work and avoids purposeless growth.

Finality. Under heavy load, Kaos runs a producer cycle of about one second and reaches finality in under 2,000 ms. This was measured on a 50-node network spanning North America, South America, Europe and the Middle East, with nodes in locations including Alaska, Hawaii, Montreal, Atlanta, Mexico City, Brazil and Israel, using the default parameters $k = 20$, $\alpha = 16$ and $\beta = 20$. Node operators can configure these parameters to suit their preferences.

2.4 Consensus and Gossip

Post-quantum consensus algorithms cannot use most traditional mechanisms, because post-quantum public keys and signatures are large and do not support algebraic aggregation such as BLS.

Problems:

- Require collecting 2f+1 signatures into quorum certificates.
- Many implementations rely heavily on compact BLS aggregation.
- All-to-all communication requires too much bandwidth.

List of post-quantum ready consensus algorithms:

- Nakamoto
- Epidemic

List of possibly post-quantum ready consensus algorithms:

- Committee
- Directed Acyclic Graph

List of consensus algorithms that require replacement or redesign:

- HotStuff
- Tendermint
- PBFT
- Others

Protocols built on quorum certificates must either carry certificates that grow linearly with the validator set or rely on signature aggregation. With 1,000 validators, a certificate of 2f+1 individual signatures is about 444 KB using Falcon-512 and 1.6 MB using ML-DSA-44, sent to every validator in every round. Post-quantum aggregation currently exists only as research-stage proof systems that add seconds of proving time per round. Snowman needs neither.

Assuming current architectures and no practical post-quantum equivalent to signature aggregation, the major blockchains that use quorum-certificate BFT protocols, such as HotStuff, Tendermint and Solana's Alpenglow, and are therefore most likely to require significant consensus redesign, validator-set reduction, or consensus replacement include:

- Ethereum
- Solana
- ICP
- BNB Chain
- Aptos
- Tron
- Sui
- Monad
- Hedera
- Cosmos
- Sei
- Movement

  Based on a gossip paradigm. In short, every node maintains a relatively small local membership table that provides a partial view on the complete set of nodes and periodically refreshes the table using a gossiping procedure. [4]

2.5 Producer Rota

Snowman decides which container the network accepts. A separate layer decides who may produce the next one. In AvalancheGo, Snowman++ schedules block producers with proposer windows. After each block, a short list of proposers is drawn by stake, each with its own window, and once every window has expired, any validator may propose. Kaos replaces this with a producer rota for two reasons:

- **No "anyone can propose" fallback.** The rota never opens production to every node. It ticks continuously through its producers, each with a 3-second window, and repeats from the start when it reaches the end. A producer that is offline, or fails for any other reason, forfeits its window to the next.
- **Schedules can't be computed in advance.** Proposer windows are seeded by block height, so anyone can compute future proposers ahead of time. The rota is seeded by the ID of the current chain-tip container, which no one knows until the previous round commits.

The rota is recalculated after every successful consensus round as a stake-weighted ordering of every staked node.

2.6 Beacon System and Proof-of-Uptime

The system deploys a unique Proof-of-Uptime mechanism. Every 15 minutes, a beacon transaction is included in the ledger by the node holding the producer rota window at that moment. A beacon transaction contains only one object, a Bloom filter [17] that contains all of the node identifiers that the producing node had active in its beacon table. The beacon protocol runs over reliable and encrypted UDP streams. It was developed for the sole purpose of being the foundation of the beacon protocol. The protocol excluding control messages only exchanges one packet, a pulse sent to all nodes when they have not been heard from for at least 150 seconds. The pulse message carries a Bloom filter of all nodes in the beacon table. If a recipient node has nodes in its table that are not in the Bloom filter, it includes those nodes in its response. This way all nodes are always learning of all nodes and maintaining a stable and accurate table. The design draws on earlier work on gossip, peer sampling and structured peer-to-peer networks [3, 5–8]. The beacon table holds all nodes in the system and each node communicates with all other nodes. The protocol is designed to handle extremely high latency [2] which makes it suitable for all types of internet connections and environments. If a node is found to be non-responding it will be probed every 75 seconds. A node is considered to be Alive if we have heard from it at least every 300 seconds. A node is considered to be Dying if it has not been heard from in 600 seconds and a node is considered to be Dead if we have not heard from it in at least 900 seconds. Finally a node after 1200 seconds will be considered Dust and evicted from the beacon table at which point a node will no longer include it into beacon transactions.

Nodes earn rewards for each beacon that lists them. The Incentive section describes how rewards are calculated and paid.

2.6.1 What a Beacon Proves

Each beacon is signed by its producer and permanently recorded on chain. It proves which nodes that producer reported hearing from. A node stays listed until it becomes Dust, 1,200 seconds after it was last heard from, so a beacon attests that each listed node was reachable at least once in the 20 minutes before it. Because beacons are 15 minutes apart, Proof-of-Uptime measures reachability at 20-minute resolution rather than continuous uptime, and a brief outage of up to 20 minutes costs

a node nothing.

2.6.2 Attacks on the Beacon System

The beacon system is not fully immune to adversarial nodes. A producer can censor nodes by leaving them out of its beacon's Bloom filter. The design limits how often this can happen, how much damage it can do, and what it costs the censor.

How often a bad producer gets a beacon. A beacon is produced by whichever node holds the 3-second rota window at beacon time and wins that consensus round. The timing cannot be planned. When the network is idle, the rota cycles through every node, so each of n nodes produces a given beacon with probability $1/n$, regardless of stake. Under sustained transaction load, the rota is reweighted after every commit and the highest-staked producers stay at its front. Either way, a single node's share of beacons shrinks as the network grows:

Nodes	Beacons per node per day (idle network)	Share of a targeted node's credit one bad producer can remove
5	19.2	19.4%
10	9.6	9.7%
50	1.9	1.9%
100	0.96	0.97%
1,000	0.096	0.1%

The last column assumes the bad producer excludes the target from every beacon it produces. The target still matches the filter by chance 3% of the time, so it loses 97% of the credit from those beacons.

Running many nodes does not let an operator choose which beacons it produces. Which node produces a given beacon depends on where the rota lands at that exact moment, so an operator with k of n nodes produces k/n of beacons only as a long-run average. It cannot target any single beacon.

How much a bad producer can censor. A beacon is valid only if the nodes it lists hold at least 80% of stake, which keeps the network live. A producer can therefore exclude at most 20% of stake, minus whatever stake is genuinely offline. At the observed 98% network uptime, that is at most 18% of stake per beacon. Censorship also earns the producer nothing: each node's reward is a share of total active stake, not of the stake a beacon lists, so excluding others does not increase anyone else's reward.

Consequences. Every beacon is signed and permanently recorded on chain, so censorship leaves an attributable history. If honest producers list a node in 98% of beacons, the chance that an honest producer omits it from m of its own beacons by bad luck is 0.02^m: 1 in 2,500 for two beacons and 8 in a million for three. Kaos has no protocol-level slashing. Enforcement happens node by node: any operator, or the community acting together, can configure its nodes to blacklist offending nodes from the beacons they produce. The trade is lopsided. An operator with k of n nodes can censor at most k/n of beacons, while the remaining producers can exclude it from up to $(n - k)/n$. A node can cheat, but it will be cheated in return, and by far more. The 80% rule applies here too: blacklisting, like censorship, can never exclude more than 20% of stake.

False positives. The Bloom filter has a 3% false-positive rate, so it credits an absent node 3% of the time. A censored node therefore keeps 3% of the credit it should have received, and a permanently offline node still collects about $96 \times 0.03 \approx 2.9$ beacons per day. The filter uses about 7.3 bits per listed node with 5 hash functions: 913 bytes at 1,000 nodes and 9.1 KB at 10,000 nodes.

Liveness. A failed beacon does not stall the system. If a beacon transaction fails for any reason, the rota keeps ticking and the next producer attempts it, until the chain moves. If more than 20% of stake is genuinely unreachable, for example during a regional outage, no producer can build a valid beacon, and beacons resume once enough stake is reachable again.

The beacon protocol is very good at determining a global network health and converging on consensus of active participants. In the first weeks of operation, the beacon protocol exposed faulty IPv6 routing at a major cloud provider, where nodes inside the provider's own network had unreliable or missing routes to one another, including between continents. We worked around it system by system, network-wide uptime reached 98%, and we have reported the issue to the provider.

2.7 Transactions, Pool and Propagation

There are two types of transactions that can be pushed into the system.

- Stake - Used to delegate a physical node into the system using its Proof-of-Possession file and an amount of credits to be locked into a bank stake. A begin and end timestamp are used to determine when the deposit will activate and deactivate the physical node and when to refund the deposit. A stake must be at least 64 credits and at most 6,400 credits. The same limits apply to delegates and delegators, and each delegate node can accept at most 64 delegated stakes. The cap keeps stake spread across many nodes, so no single node can dominate production, beacons or rewards.

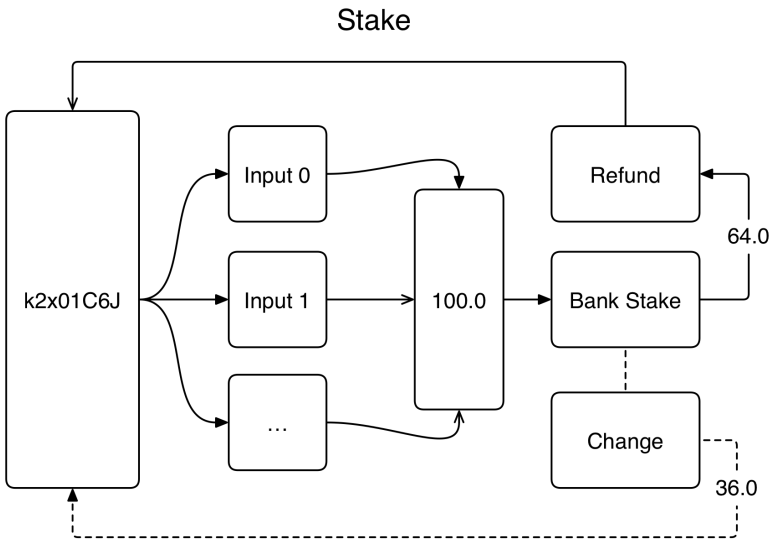


Figure 1: Stake transaction. Inputs fund a bank stake; change returns to the sender, and the stake is refunded when it expires.

- Transfer - Combines any number of inputs into at most two outputs: the payment to its destination at index 0 and the sender's change at index 1. The two-output limit prevents

batch payouts. When one transaction pays many recipients, as exchanges commonly do with withdrawals, it permanently links every recipient's output to all the others. If any one of those outputs is later tied to illicit activity, the rest are associated with it on chain. Because every Kaos transfer pays exactly one recipient, a user's coins are never linked to unrelated recipients through a shared transaction.

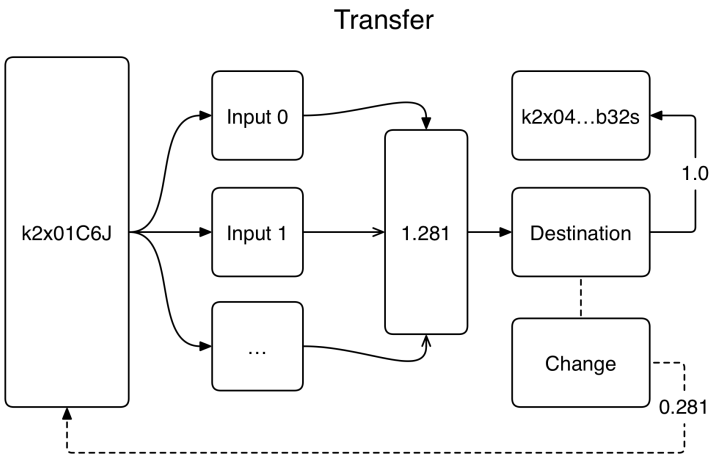


Figure 2: Transfer transaction. Inputs pay a single destination at index 0, with change returned to the sender at index 1.

The system uses a global transaction pool with an efficient gossip mechanism to propagate transactions. Nodes periodically share their transaction pool contents via Bloom filter to other nodes and exchange transactions that do not match the filter. This design ensures the system cannot be flooded with transactions at an uncontrolled rate while also ensuring that producers are able to access transactions quickly.

2.8 Fees

Every transaction pays a fee, and all fees are burned. Fees are denominated in units, where 1 credit = 1,000,000 units. A transaction's fee is the sum of three parts:

Component	Fee
Base fee	100 units (0.0001 credits) per transaction
Keying material	10 units (0.00001 credits) per full 1,000 bytes of public keys and signatures
Amount	0.01% of the amount sent

There is no per-input fee. Small stakers receive many small reward outputs and later need to combine them, and a per-input fee would make that consolidation a recurring cost for them.

The keying-material fee scales with the size of the signature scheme, so larger keys and signatures cost more to transact. For a single signer:

Scheme	Public key + signature (bytes)	Keying-material fee (units)
Falcon-512	1,649	10
Falcon-1024	3,255	30
ML-DSA-44	3,732	30
ML-DSA-65	5,261	50
ML-DSA-87	7,219	70

For example, sending 1 credit (1,000,000 units) from an ML-DSA-44 key costs 100 + 30 + 100 = 230 units, or 0.00023 credits.

2.9 Incentive

Kaos issues new credits only through the beacon system, as a subsidy paid to stakers for keeping their nodes reachable. There is no per-container producer reward.

Annual emission. The amount issued each year depends on the total supply. Emission starts at 6.4 million credits a year and is cut by 1.28 million for each full 6.4 million credits in circulation, until it reaches a permanent floor of 640,000 credits a year:

Total supply (credits)	Annual emission (credits)	Per beacon (credits)
under 6.4M	6,400,000	182.65
6.4M to 12.8M	5,120,000	146.12
12.8M to 19.2M	3,840,000	109.59
19.2M to 25.6M	2,560,000	73.06
25.6M to 32M	1,280,000	36.53
32M and above	640,000	18.26

Kaos therefore has a permanent tail emission rather than a fixed supply cap. All transaction fees are burned, so net supply growth is emission minus burned fees and unissued shares.

Rewards per beacon. Each year's emission is divided evenly across 35,040 beacons (96 a day for 365 days). Every node listed in a beacon receives the per-beacon amount multiplied by its stake divided by the total active stake across the network. A node that a beacon does not list, because it was unreachable or excluded, receives nothing for that beacon, and its share is never issued. The average yield across the network is the annual emission divided by the total stake: with 6.4 million credits emitted and 10 million staked, for example, it is 64%.

Delegate commission. A delegate, the operator running the node, receives a fixed 6.4% commission on the rewards earned by stake delegated to its node. The rate is built into the chain's consensus rules, so no delegate can raise it, lower it to undercut others, or change it after delegators have joined. Delegators choose a node on uptime and reliability alone, never on fee terms.

Payout. Rewards accrue per beacon and are paid every 96 beacons (24 hours). A new stake forfeits its first 96 beacons: their rewards are minted and burned rather than paid, a 24-hour warm-up. When a stake expires, any rewards it has accrued are paid out. Because rewards for warm-up beacons are burned, supply growth falls during periods when many new stakers join.

2.10 Virtual Transactions Per Second

Transactions per second (TPS) counts transactions. That works when the chains being compared use transactions of about the same size, and classical chains largely do: their transactions are built around a 32-byte public key and a 64-byte signature. Post-quantum key-and-signature pairs are 17 to 75 times larger (1,649 to 7,219 bytes, against 96), so a single post-quantum transaction carries as much data as many classical ones. Comparing a post-quantum chain with a classical chain by TPS therefore compares unlike units.

Kaos measures throughput with virtual transactions per second (vTPS). The purpose of vTPS is to express a post-quantum chain's throughput in units of classical transactions, so that Kaos can be compared directly with pre-quantum chains. vTPS is computed for every container by the engine and reported on the network dashboard.

Definition. Each byte of a container is weighted by what it carries:

- **1** for header fields: chain ID (2 bytes), height (8), parent ID (32), timestamp (8) and size (4), 54 bytes in all.
- **8** for the container's object, which carries the transactional data: inputs, outputs, keys and signatures. Transactional data is weighted more heavily because it must be transmitted, verified and stored, while header fields are mostly transmitted.

One virtual transaction is 2,048 weighted units, which is 256 bytes of transactional data. That is roughly the size of a typical classical transaction, so one virtual transaction corresponds to about one classical transaction's worth of data. For a classical chain whose transactions average 256 bytes, vTPS equals TPS.

vTPS is the container's weighted total divided by 2,048, divided by the minimum container spacing in seconds:

$$\text{vTPS} = \frac{\text{header bytes} \times 1 + \text{object bytes} \times 8}{2048 \times \text{spacing}}$$

Because it uses the minimum spacing, vTPS reports the rate a container's contents would sustain if containers were produced back to back, which is the network's peak throughput at that load.

Example. Both systems below produce one container per second.

Kaos, a container of 64 transactions signed with ML-DSA-44:

- Header: 54 bytes × 1 = 54
- Object: 64 × (1,024 bytes of inputs and outputs + 3,732 bytes of public key and signature) = 304,384 bytes × 8 = 2,435,072
- Total: 2,435,126 ÷ 2,048 = **1,189 vTPS**, from **64 TPS**

Classical chain, a block of 640 transactions:

- Header: 72 bytes × 1 = 72
- Transactions: 640 × (1,024 bytes of inputs and outputs + 96 bytes of public key and signature) = 716,800 bytes × 8 = 5,734,400
- Total: 5,734,472 ÷ 2,048 = **2,800 vTPS**, from **640 TPS**

By TPS, the classical chain appears 10 times faster than Kaos. By vTPS, it moves 2.35 times as much transactional data. The difference between those two ratios is the size of post-quantum signatures, which TPS does not account for. The remaining 2.35× gap reflects the block sizes

chosen for this example: 717 KB for the classical block against 304 KB for the Kaos container.

The network dashboard shows vTPS by default and switches to TPS with a click, so both figures are always available. TPS gives the number of payments settled; vTPS gives the volume of transactional data processed, in units a classical chain would recognize.

3 Kaos Protocol

The protocol runs exclusively over IPv6, on two transports: peer sampling runs over TCP, and the beacon protocol runs over UDP. Both perform the same key encapsulation, encryption and credential exchange. The protocol provides post-quantum key encapsulation, strong symmetric encryption and replay-resistant peer authentication. It ships with a default set of algorithms, and new ones can be added easily.

Kaos uses ML-KEM alone rather than a hybrid with a classical key exchange. A hybrid's classical half offers no protection against a quantum adversary, and Kaos's design goal is that no layer depends on pre-quantum cryptography.

3.1 Network Topology

The network has three kinds of participants:

- **Delegates** are the staked validator nodes. They connect to one another, produce containers, vote in consensus and take part in the beacon system.
- **Interface nodes** serve RPC and WebSocket connections. They connect to delegates and give clients a way to read the chain and submit transactions.
- **Clients**, such as wallets, connect only to interface nodes.

Delegates should not serve RPC. They can, but opening a validator to public RPC traffic exposes it to attack, so Kaos keeps the two roles on separate nodes.

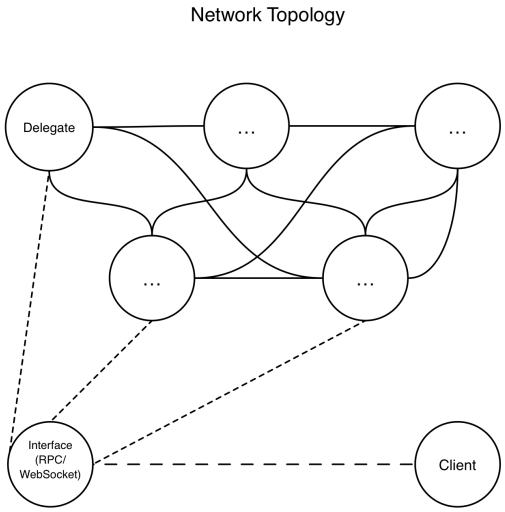


Figure 3: Network topology. Delegates form a mesh; clients reach the network through interface (RPC/WebSocket) nodes.

3.2 Key encapsulation mechanism

The system deploys three default key encapsulation mechanisms [12].

- ML-KEM-512
- ML-KEM-768
- ML-KEM-1024

3.3 Algorithm Selection

For each new session, the initiator picks the ML-KEM parameter set and the cipher at random from the supported options. Every option meets Kaos's security floor, so the weakest possible session (ML-KEM-512, NIST security category 1) is still post-quantum. Random selection also spreads sessions across algorithms, so a weakness later found in any one of them would expose only a fraction of sessions.

This suits a peer-to-peer network, which cannot respond to a broken algorithm the way a centralized service can, by switching every server at once. Instead, a network upgrade removes the affected algorithm from the random picker, and new algorithms are added the same way. Kaos never negotiates down to a weaker algorithm: if a peer does not support the one chosen, the session fails and is retried by either end.

3.4 Authentication

The base protocol establishes an encrypted session without reference to node identity. Identity is proven on top of that session. Each peer presents signed credentials containing a challenge nonce, a BLAKE3 [15] digest of the current session, its software name and version, its node ID and its host name, all signed with its post-quantum validator key. The nonce prevents credentials from being replayed. The session digest binds the credentials to the session they were issued for, so credentials relayed from a different session, for example by an attacker sitting between two nodes, fail verification. A session is attributed to a peer only after its credentials verify.

3.5 Encryption

The system deploys two default encryption algorithms, both operating with 256-bit length keys.

- AES-256-GCM
- ChaCha20-Poly1305

3.6 Compression

The system deploys a single compression algorithm, applied to stream data before encryption. LZAV [16] was chosen as a fast general-purpose in-memory data compression algorithm. LZAV is safe even when decompressing malformed or damaged compressed data and does not require calculation of a checksum on compressed data. This makes LZAV an excellent choice for high-bandwidth network streams.

4 Kaos Application

4.1 Key Diversity

The Kaos engine was engineered from the start to support any number of digital signature algorithms. Users can choose their preferred key types or select them based on use case. For instance, a cold wallet would be more suited to an algorithm such as Falcon-1024 or ML-DSA-87, a hot wallet would be more suited for ML-DSA-44 or ML-DSA-65, a donation wallet which receives lots of transactions but also maintains a small balance would be more suited for Falcon-512.

Different keys come at different costs both economically where larger signatures are more costly to transact and conceptually where they represent different tradeoffs between security, speed, and privacy.

4.2 Address Format

Addresses are built on BLAKE3 hashes [15]. An address such as k2x01C6J9uN5PX1k8TDyhk xGCz4pXZ58WByuhzbd28qc8mrpz3pWG, a stagenet address shown for illustration only, has a five-character header followed by a Base58-encoded body, 53 characters in all:

Part	Example	Meaning
Network identifier	`k`	Kaos
Deployment	`2`	the network deployment the address belongs to (here, stagenet)
Marker	`x`	separates the deployment from the key type
Key type	`01`	the signature scheme of the key, in hexadecimal
Body	`C6J9uN5P…`	35 bytes: a 32-byte BLAKE3 hash of the key type, version, deployment and public key, followed by a 3-byte checksum of that hash

An output can be spent only by a key whose address matches it exactly, including its deployment, key type and version. The format provides built-in versioning, integrity checking, domain separation and compact size. A single altered byte in the body fails the checksum, and because the checksum is carried in the body, the body remains independently verifiable with the header removed, offering even greater compaction. The key type, version and deployment are hashed together with the public key, so the same key produces an entirely different address on each deployment and version. A key can be used on any deployment it is initialized for, and an address from one deployment or version can never be mistaken for one on another.

Addresses use the full 256-bit BLAKE3 output. Finding a different key with the same address would take about 2^{128} Grover iterations, each a full post-quantum key generation, which matches the strongest key types Kaos supports, Falcon-1024 and ML-DSA-87. The address is therefore never the weakest link, whichever signature scheme a user chooses.

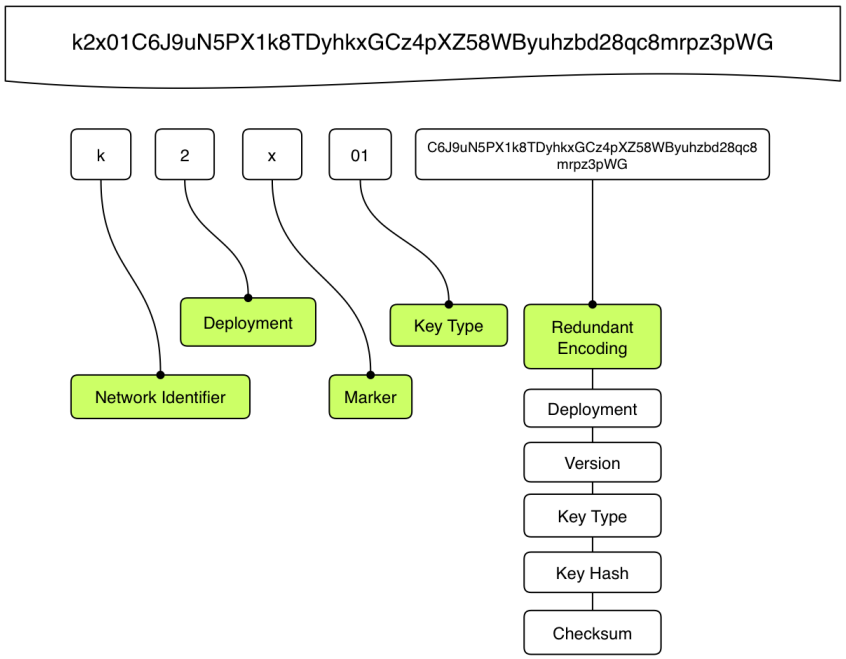


Figure 4: Address format, shown with a stagenet address.

4.3 Wallet Encryption

Wallet files are encrypted with AES-256-GCM under a key derived from the user's passphrase with scrypt ($N = 2^{19}$, $r = 8$, $p = 1$, 32-byte key). The 256-bit key keeps a 128-bit security margin even against Grover's algorithm, the best known quantum attack on symmetric ciphers, and scrypt's memory-hard key derivation makes guessing passphrases expensive on any hardware.

4.4 Goodbye Seed Phrases

A seed phrase concentrates an entire wallet into a single secret, written down once and kept in one place. Anyone who sees it owns the funds. That single point of failure is exploited constantly, through phishing, malware, and compromised hardware. In October 2026, tampered Ledger hardware wallets sold through an authorized reseller were found to contain a hidden LTE module that read new recovery phrases from the screen during setup and sent them to attackers over the cellular network. More than $80 million was stolen within a week [10].

Kaos does not use seed phrases. Most seed-phrase wallets rely on BIP-32 hierarchical key derivation, which depends on elliptic-curve algebra that post-quantum signature schemes do not have. Rather than modify post-quantum schemes to fit, Kaos protects private keys with Shamir's Secret Sharing [9].

Shamir's scheme uses polynomial interpolation to split a key into n shares, any t of which reconstruct it. Mobile wallets default to 2-of-3: three shares, any two of which recover the key, so one share can be lost or stolen without losing or exposing the wallet. We chose a small default deliberately, because secret sharing is new to most users and managing many shares adds friction. The command-line client, built on the same library, lets users choose any t and n the share format supports. Fewer than t shares reveal nothing about the key, regardless of the attacker's computing power, classical

or quantum: the scheme is information-theoretically secure. Shares can be stored in separate places or held by separate people, so no single theft, loss or compromised location exposes the wallet.

Secret sharing protects keys at rest. Like any wallet, it still depends on the device that generates the key and later reconstructs it being trustworthy.

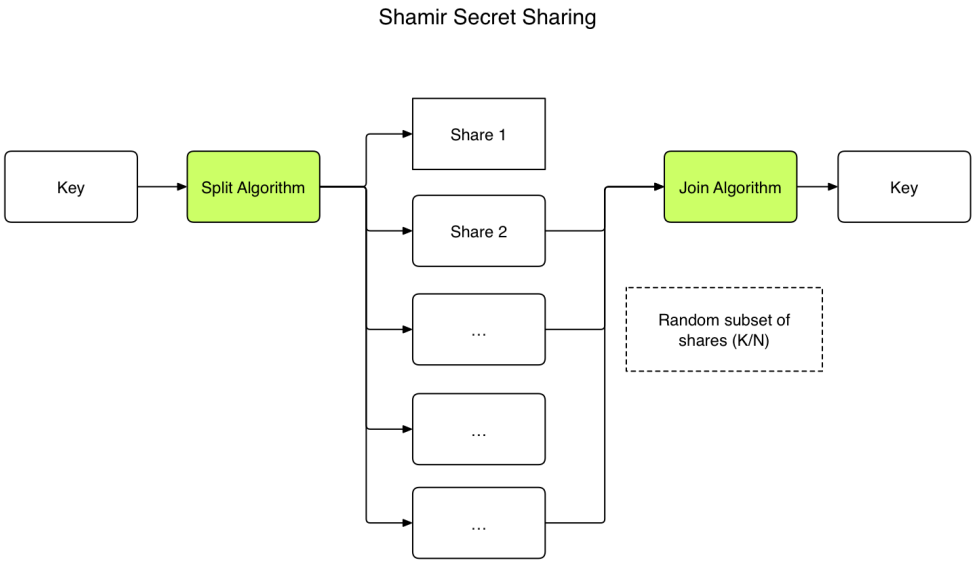


Figure 5: Shamir's Secret Sharing. A key is split into n shares, any t of which reconstruct it.

The same technique is used to protect other high-value, long-lived secrets, such as certificate-authority root keys.

Acknowledgements

We thank the anonymous reviewers and testers for their participation and input. Kaos also builds on decades of work in cryptography, distributed systems and digital cash, and we are grateful to the researchers and builders whose ideas made it possible, including Matthew Allen, Ittai Abraham, Adam Back, David Chaum, Cherry, Cobie, Frank Dabek, Wei Dai, Peter Druschel, Guy Golan Gueta, Wonho Kim, Sunny King, John Kubiatowicz, Mohit Lad, Kevin Lewi, Dahlia Malkhi, Ilya Maykov, Satoshi Nakamoto, Scott Nadal, Ricardo Oliveira, Nouha Oualha, Rama, Michael K. Reiter, Yves Roudier, Andrey Sabelnikov, Claus P. Schnorr, Kevin Sekniqi, Niklas Sinclair, Ion Stoica, Nick Szabo, Nicolas van Saberhagen, Stephen Weis, Maofan Yin, x281x1, Ben Y. Zhao and anyone else we may have forgotten.

References

1. Team Rocket. Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies, 2018. https://gitea.com/kaos/Team-Rocket/raw/branch/main/bafybeidcpizm6szhtthry3ldmsc3u5edq4hlu2p2kvgl7l6ja32miy5syi.pdf (also arXiv:1906.08936, https://arxiv.org/abs/1906.08936)

2. WonderNetwork. Global Ping Statistics: Ping Times Between WonderNetwork Servers. https://wondernetwork.com/pings (accessed 2024).
3. J. Kubiatowicz. Extracting Guarantees from Chaos. Communications of the ACM, 2003. https://oceanstore.cs.berkeley.edu/publications/papers/pdf/CACM-kubiatowicz.pdf
4. M. Jelasity et al. Gossip-based Peer Sampling. ACM Transactions on Computer Systems, 2007. https://www.distributed-systems.net/my-data/papers/2007.tocs.pdf
5. S. Boyd et al. Randomized Gossip Algorithms. IEEE Transactions on Information Theory, 2006. https://web.stanford.edu/~boyd/papers/pdf/gossip.pdf
6. PEERSWAP: A Peer-Sampler with Randomness Guarantees, 2024. https://arxiv.org/abs/2408.03829
7. C. Gkantsidis, M. Mihail, A. Saberi. Random Walks in Peer-to-Peer Networks, 2004. https://www.math.cmu.edu/~af1p/Teaching/INFONET/Papers/P2P/Mihail1.pdf
8. Chimera: A Library for Structured Peer-to-Peer Application Development, 2006. https://gitea.com/x281x1/Chimera/raw/branch/main/chimera.pdf
9. A. Shamir. How to Share a Secret. Communications of the ACM 22(11):612–613, 1979.
10. A Spy in Your Wallet: Tampered Ledgers Drain $80 Million. Weiss Ratings, October 12, 2026. https://weissratings.com/en/weiss-crypto-daily/a-spy-in-your-wallet-tampered-ledgers-drain-80-million
11. Ava Labs. AvalancheGo. https://github.com/ava-labs/avalanchego
12. NIST. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard, August 2024.
13. NIST. FIPS 204: Module-Lattice-Based Digital Signature Standard, August 2024.
14. P.-A. Fouque et al. Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU, Specification v1.2, 2020. https://falcon-sign.info
15. J. O'Connor, J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn. BLAKE3: One Function, Fast Everywhere, 2020. https://github.com/BLAKE3-team/BLAKE3-specs
16. A. Vaneev. LZAV: Fast Data Compression Algorithm. https://github.com/avaneev/lzav
17. B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM 13(7):422–426, 1970.

Author Key

The following is the authors' ML-DSA-87 public key (2,592 bytes, hexadecimal). The authors can prove they wrote this paper by signing any challenge message with the matching private key.

```
9c7de22b3c2ad76b4bac03327fec04aab0e11cd0c6aa62f95a7c0ce294062b6c
75f6eaef08ed131dda657454f70e08c6c204726a819b13f91d3d12c08a59a7fe
b8dfc4d03a9f60a4f8dcc05d3dee9a0327dbcb855afaa9afe9d5362294cb6f11
af2309cf82871a5fcde87ef1666678a9fa52b9e328475e9dc6c9a74e2cb5346e
b218553102792de35b1787152d2c61072e91625c40366d522cbe3666aebba4c5
c9c6becea3895996344ca2e76fc30bf250349cdbd0571bc8d32688036ff4a889
b2f5637acbb151de7af588f1c7f42d53c5e16533018061173c2447c11efde5a8
4853f019e19b1334f873a6c0c9a6eaa2afad74fb38daa948000527c8740edcba
1715026042ec48c34905c944eaf13855c75912c9f8c0fa77434af205521af88a
5fef936071afed712efc0e90a4b7921d4d1b6c7aaf73dedfc1e184cf3efded7a
bfd30006cbb0c4d064754e3a910c08adafb5ac658b1cffe6b9a8d239fa63c1a8
af7eb5b2e732284ea902f99a0d62da12da79cc6e31a1b933c5c7e5a25534cb98
e411e77edeb012425932fbd1f66b190001efdb94b9be8f98408153a152aa9f01
71df1ca5d31ea12e5357d31a88846f2211babee21dfbff2ee613c72a9b484ae8
c2a2438f00714de1316fc5613418aa24455a05f7003ba2d50cde8c7cd6c20c9d
74deabfa3f29dba3a11806370470eeffd4d175876008c88bccf65d19a0098de9
56526fc640f6e6299c8073cb207e11054f27331e40b5eac250b5267ebf04c233
bdefcfdd6f752212776bbefdd60749794eae1e6ae48c37d8a5b73dd0f4ddd28c
4f8cd5a404904831c8d9fd5228796dfc5ad4979c16a7c3487e0944c325a76343
```

```
a25d7e1da1389d1658a48687b86f6d1288563f22874d5264fb5a65c5dbd0d042
b5444e6560e53ea49ac8cc2ce2d43765ce35c82583dc699caf9f3493aace54fc
65ca8c4a525a212a52c09d11f529a3174a49ca9f0489034e52d180f8c4f34abf
b2182ecbea41188f08fa7188d87fe34486c92fd70901cb398e7c456e8ac998ef
c81068d5785e146dbf6e1847eaf73e1d274344f2c0f3d08ca7ba1b1a7fa3c034
a5bba26d9afc63a036c33d0c48a1dd3928a9c82b47fe8857eebb3d2fbee1b9ef
d1a0076684d7121a36d9872abc88f2ba701ba40394ee7ebc526cc0be11bc3f9d
5636cbab87ce7743c2c421d168c173cd1c1224fde9d4bf640bfe700b85cec352
cb32955ce45babfae858b5e1d494be06a05ba135d67a2540d0405477adb316e2
108a7c12e8e5e1a75a31d4c29d33072fba0e830408249011909e6c7dff8e51db
8b59556c137a8ec82851e2817cc70704ccad36461c621942e7ec5a27262f8309
f27b22797b1f0500dbf61865baefa20d2911b169d50fe27a3a4412345eee31e5
e2e8534da9159052cc0d91ebd5d419923c5daf8c9880da615715757a1cd35312
22564aaaf340e668e33e6f59baa83be125bfdb5084b1e1383558e1e750bbde00
46a15897534875f3d654ba5af9576f5ed77c1ea9afba33a10729d100eefece7c
7b31174a45f1401a6c59773e978636891fe6bb1aa29ec9d5e7a0b74b2437c7b3
a1f28fe10278783bea267c280036d37a78f9b102cf6d5abba6692e2725809896
3acc6c1daeb7e310b3e1deb9470207951c54cbb1769fc7974443ea40911f3466
b4fbd13fa597ec3e42fa0d2808424d60e7902d8dea7f8f1ca137d329d2731860
b1cb870b717ae298220c7d01660b3444a77878579f83803210d6c19c03a52a45
60c18a578cd0aad1cdc546a1935fbcd7f659064dca53d81013051f4c698588b3
83ea6ee4e81df73b5348a3209d5632f70afb941aa12d145a18a749ecd6810591
dd43f551c7d46f2fe037dc93879995d6b8e0eb9097db240f054aa9eb48f4e1b2
879496ed5528b81d1da0b9ba483e434028c8a7820085575093fdfc0e6fcae632
62484f2b8354234f72ecea77147249d9a925428b30422013d73c7f4464312d67
693a0441f6f40e7631694758479a6d9a051f4be99356cefed56e693a55d9bbcd
c44e7446abbf1c6130d6dc5ac0bb329ab776b441f7ad452fbdafb4d918f9d9d7
e55abb7d4fa68c1e518c72fd03927aecea9fe3ef6cf53bde655ebac5094b15e2
dd07620bdff51ed399cda70450e2dcadc319b24a3946a6b633d0425348a7047a
e5d44a0c33581ae6a32e29003b8c323c9d82c826e7a39d9484bd3a19d1f0bdc9
a8e40c3806dddd7e16988a6a8cb959229c1b5d1a8e69a4d8cc9de5abec551882
2ee54bf9dcfa92a3465321f0d6f7d0c15410fc53ed4896da0dc7f871610c55a4
97ff898e3bd2b9df54ca9392c13311c87449d2cf98cc9fb6051f85ae92999449
32b655aaf01cc3cae3b505e258b8ea14d8f608208f098c2dc894bbbd3d2793ab
ff67e5ae1ca2a27b8f872f0caff86b6c7c38006c8088670cbd160a63e802c805
a05f8ab4a8c947149262c088d8eb2812ac24214eb95118bd6293ee340d7dfaba
6cb5f095630e8e98eda1cf16225bea071d887b6318539894669c0a001f1c6ddb
cb95893f03db31fc82b504463ba1e7686090b7c07e19cd99eefeed3d79d97f90
ff8bbba972323b3718ca874c44a421fb066999479c87114a77c76516e50e13ea
ef2bcac5828353295a719b643d71927707f0b121a175f7faa8799202b66567ba
bc6de5b103f8c7330527bd9a25a410a34674f2c1bacfca04f0fa098077ce614e
3fff431a250e4c05cdc747518642a2afe470395ec49ae3be113cfe91330369f0
505197bb15da8c7758294353e9ccab99750729bdf140492f64c1062621d16cf1
973604c42df7e68fb1bfa2274b1fbc1899224195598a5e88c22f53dd3b533de0
6db6e0013a2ed2ebec687813818cb68426abee632a8f94bd5e044f3367aaf7bd
9b32a88ac19b3b7da7c63de3ab5749494c2f36cda29ac87aae2308361ee70038
cd17b01f34ea7b67a1bdcbf90c1c264958fcdccfa002de1c9b0e1bc418af6713
f43484ff75a029efda44e517d3712f406194a6cae372968b1afb1d895294099a
649b4e3df3fd31969e5acee66be8240732da0c360a359428d1de41481c18faf7
c696eae0b7c67721316aceaa062b6ff4088d2a22ea3e54bd0608505529129c02
955cf787e9ec670601c51bba90614cd356cb95ba98028b816c886ef8d6a752de
f057041f02c0f7d5eda5a95d92b2582fb8e0b308320b80fb1a4b69b9608762e0
0d8be33e5b5015f300f21152f9f8301ae45eba638d8ab4dd2602edfef8925597
3b9e98c2a1a2c923c4ade85fa40153d69e907589ef7de67b54f8fc191df0d220
08479c0ee6ffeaa2c557f98690ca76534d599f5a371cbd041335999abb33c5ea
228b3e93f5191bbc77d9588e80e9169f2918410e3774332569f56d6450d847fb
bc261d0ffb8066597b51e484d49facbea2da8e4e26022ade66d16a77d11f4da8
578c72c0022da23687069497f0d3dc223a7ee4c79c85c6b938e9ddb19f0ed8bb
3413ab57661d623bf9140337541f97c8a2fcd5462e6afbe34eea38e8fc801574
4f7c1567e706edd79ed72aad9719bc062d9c9b37b7347db4863bb09ae3eb45ba
4af8abb7e2c052184ca309104de12d108e33feb25e54d476c4181bdb3a2882bd
51e4206a92aeb5e01c228f564d90074ae8399f73e230a35f54a62d1a485b12d0
```