Joltzu: A Safety-First Blockchain for the Real World

Sam Witch

Working Draft - Revision 2
5 November 2026

Abstract

Joltzu is an EVM-compatible, proof-of-stake Layer 1 blockchain which makes user protection a part of the protocol. It uses post-quantum lattice cryptography, native smart wallets, shared liquidity among dapps, a purpose-built Joltzu Wallet, a 3D interactive world, a security council, circuit breakers (cap how fast value can leave protocols), account safeguards (make large withdrawals and policy changes wait so they can be cancelled), contract invariants, a native perpetual orderbook, lending, oracles and plugins. It pushes native smart contract execution into an optimized C++ node targeting one million transactions per second and sub-second finality with 500 ms slots.

There are two parts, one which is the protocol itself, handling decentralized consensus, transaction execution and native services, and another which is the Joltzu Service Network (JSN), handling off-chain tasks and AI-driven contract screening. It complements the protocol with purpose-built private RPC and indexer hosting, a post-quantum VRF for on-chain randomness, keepers, a co-signer that only approves transactions meeting the owner's rules, a signature verification network and AI-driven contract analysis.

Joltzu uses sharded, stake-weighted BFT consensus, where validator groups certify batches for disjoint account ranges in parallel and a global committee then commits the certified batches, in a deterministic order, into one block. Validators sign with ML-DSA, a post-quantum scheme, and any account can add an ML-DSA key without changing its address future proofing the chain against rapidly evolving AI-driven quantum computers.

1. Introduction

Most losses on public blockchains have nothing to do with broken cryptography. They come from a user approving a malicious contract, trusting a lookalike website, losing a key or using an application with an exploitable bug. Binance's initial investigation traces the largest theft on record, about $6.4 billion in October 2026, to one engineer's compromised laptop [1]. Each wallet and application defends against these risks on its own, which means a single mistake usually leaves the user with no protection at all. Section 7 shows how this pattern produced the largest thefts to date and why AI is making it worse.

Existing wallets do too little to prevent these losses and sometimes it's just not possible. Hardware wallets for instance still blind-sign requests that their owners cannot read, which is how Bybit's signers approved a $1.46 billion theft [2]. Once a transaction is signed no wallet can slow down the funds leaving. Wallet drainers alone took $494 million in 2024 [3]. This is why Joltzu ships its own wallet, Joltzu Wallet, built around the protections existing wallets lack (Section 8), and backs it with protocol rules and support for post-quantum accounts.

Common financial services, including markets, lending, payments, escrow, subscriptions and inheritance, are native to the chain, so developers can compose them instead of rebuilding them and users deal with one consistent permission model.

Developers can keep using the existing tools developed for the EVM [4], such as Remix, Hardhat and ethers, and apply similar techniques to build on the chain. JOLT is the common asset for fees, staking and applications, and any additional ERC-20 asset needs Security Council approval, with an emphasis on tokenized stocks, verifiable claims and payment assets. The Lab funds useful applications and sustained development, so a developer can build a business without launching another token.

1.1. Two parts: the protocol and the Joltzu Service Network

Joltzu has two parts, and its applications, including Joltzu Wallet, the developer platform and Joltzu World, build on both (Figure 1). The first part is the Joltzu protocol, the chain itself, which covers consensus and staking, execution and accounts, authenticated state, safety rules and native

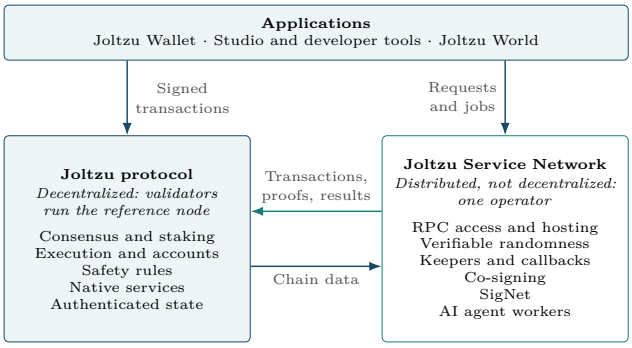


Figure 1: *Joltzu's two parts. Validators running the reference node operate the protocol, and the protocol alone decides the ledger. The Joltzu Service Network does the off-chain work for applications, but consensus never depends on it and validators recheck whatever it submits.*

services. This paper specifies the design, and the Joltzu documentation tracks what is implemented. The protocol is decentralized because stake-weighted validators run consensus and every validator executes the full ledger.

The second part, the Joltzu Service Network (JSN), provides the off-chain services: public RPC endpoints, application hosting, verifiable randomness (VRF), keepers, co-signing, SigNet and the AI agent network's workers. JSN is distributed, not decentralized. Many machines provide availability and capacity, but one operator runs them all. JSN runs separately from the validators and consensus never depends on it, so an outage can delay its services but not block production or finality. Everything it submits is an ordinary transaction or a signed or proved result, which the protocol rechecks. Anyone unwilling to trust the operator can verify state and receipt proofs, run their own node, host an application elsewhere or, where the protocol allows, execute a due payment themselves.

1.2. Design principles

We designed Joltzu around four principles, and they come up throughout this paper.

1. **Protection in the protocol.** Validators enforce account policies and safeguards, outflow budgets, holds and locks on every spending path, so it doesn't matter which wallet someone is using.
2. **Separated, bounded authority.** The security model relies on committees with limited, well-defined powers. Committees can only act with a supermajority of their full membership and leave a public record (Section 5.6).
3. **Contain without halting.** Circuit breakers cap how fast value can leave (Section 5.2) and bonded pauses and Council holds stop the funds involved (Section 5.7). The network never gets paused and finalized history is never rewritten.
4. **One shared ledger.** Native services share balances, reservations and settlement, which means you could lend funds, add orders to the orderbook and many other things simultaneously.

1.3. Organization

We start with the protocol core in Sections 2–4, which cover consensus, execution, accounts, state and the validator node. Sections 5–7 then explain how the safety rules and the AI agent network prevent and contain losses, and test that design against the largest exploits to date and the growing threat from AI. Section 8 covers Joltzu Wallet, and Sections 9–12 the native services, which are payments, conditional transactions, identity, markets and lending. Section 13 describes JSN, and Sections 14–15 the developer platform and Joltzu World. We close with governance and attacks, and the appendix gives the TLA+ specification of consensus (Appendix A).

2. Sharded Consensus

The issue every distributed system faces is how to agree on a value among a set of nodes. Joltzu's validators run the reference node and do this through stake-weighted Byzantine fault-tolerant (BFT) consensus, secured by the JOLT that they and their delegators stake. To spread the work of collecting and certifying transactions, validators are split up into groups, each responsible for a range of account addresses, and these groups prepare and certify batches concurrently. Global consensus then commits a Jolt block, which is one ordered list of certified batches that every validator executes against the shared state (Figure 2).

Groups divide up the consensus work and not the state. Every validator holds and executes the complete ledger, so there are no shards to bridge between and an application sees one state, whichever groups its users belong to.

2.1. Account ranges, epochs and committees

Joltzu keeps the 20-byte EVM address space $\mathcal{A} = \{0, \ldots, 2^{160} - 1\}$. With N groups, a transaction from sender address a is routed to group $\lfloor aN/2^{160} \rfloor$. For instance, two equal groups divide the space as

$$\mathcal{A}_0 = \{0, \ldots, 2^{159} - 1\}, \tag{1}$$

$$\mathcal{A}_1 = \{2^{159}, \ldots, 2^{160} - 1\}. \tag{2}$$

Routing uses the sending account's own address, even for a multisig account, and its signers' addresses play no part.

Time is divided into epochs of 512 slots, where each slot targets 500 ms. This means that an epoch lasts 256 seconds (just under 5 minutes). Each epoch has a fixed committee and group assignment, and a block may announce the committee for the next epoch. Once that block is finalized the change takes effect at the epoch boundary, and followers and light clients learn about it from certified history. Since each group signs under its own domain, a certificate cannot be replayed in another group or epoch. Figure 3 shows an

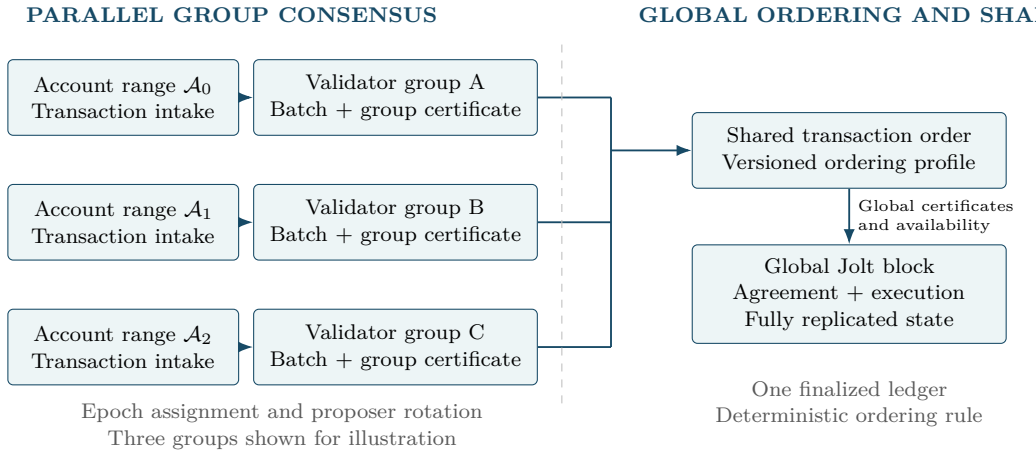


Figure 2: *Groups certify batches at the same time, then global consensus commits to one transaction order and every validator executes the same Jolt block against the shared application state.*

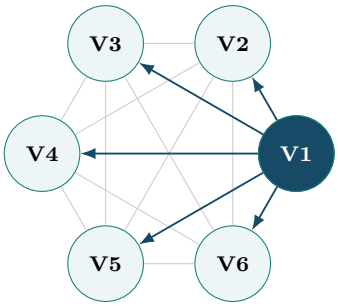


Figure 3: *An illustrative group of six validators. The highlighted arrows show a proposal being sent out, and the peer links also carry votes and recovery messages. The actual committee size will come from security analysis.*

illustrative group of six validators, but production committee sizes will depend on stake distribution and adversarial analysis.

2.2. Randomness and proposer scheduling

RANDJAO orders proposers within each epoch and seeds the assignment of validators to groups. Each validator commits to a Merkle root over secret values, one for each height, and the root and tree depth are part of its committee record. A proposer includes a *reveal* in its block, which is the secret for that height along with its Merkle path. Because the root fixes every secret in advance, a validator has exactly one valid reveal for each height. A signature couldn't give us this, since ML-DSA signatures aren't unique and a proposer could keep signing until the output suited it. The randomness of a block is the hash of its parent's randomness and the new reveal, and block headers, finality records and checkpoints all carry this value so that light clients can verify it. Checking a reveal takes a few dozen hashes and needs no state. A tree of depth 28 lasts about 4.25 years at two blocks per second, and a committee transition can install a fresh root before the heights wrap around. The proposer order of a new epoch is then seeded from the randomness of the block that announced that epoch.

A proposer can still bias this chain of reveals by withholding its block, and the bias reaches the epoch seed which sets proposer order and group assignment. A threshold BLS beacon would remove that bias, but it wouldn't survive a quantum computer and there is no standardized post-quantum equivalent, so we don't use one. The proposer gives up its block to do this, and we accept the remaining bias as a limit of the current design (Section 17.5).

2.3. Group certificates

Each group queues transactions from its account range and forwards them to the scheduled leader, and batches respect sender nonces, fee priority and resource limits. Members validate a batch and sign votes bound to its proposal with ML-DSA-44, a post-quantum signature scheme [5], and a certificate lists the signers' addresses with one signature from each. These signatures don't aggregate, so a certificate grows by about 2.4 KB for each signer, and we cap an active committee at 32 validators (Section 16.2). Every validator key is registered with a proof of possession, which shows control of the key before it can join. For total assigned weight W, a certificate needs a set of signers Q whose weights w_v satisfy

$$\sum_{v \in Q} w_v > 2W/3. \tag{3}$$

Offline validators still count towards W, so an outage cannot lower the bar for the validators that remain. This means that committee security depends on voting weight and on how committees are sampled, and the raw number of validators isn't what decides it. A certificate shows that a group validated a batch, but it doesn't finalize state or guarantee that every call in the batch will succeed.

2.4. Global ordering

Global consensus commits which certified batches are included and the order in which their transactions execute. We chose a deterministic ordering rule that ignores arrival time. Transactions are ordered by gas price, although a sender's transactions always stay in nonce order. If two transactions share a sender and nonce, only the higher-paying one is kept, and any remaining ties are broken by transaction hash. A transaction whose declared gas no longer fits in the block is deferred, together with its sender's later transactions. Commit evidence binds the parent block, epoch, slot and ordered transactions.

A call in a certified batch can still fail if an earlier transaction changed the state it relied on, because shared state is resolved in this global order. Gas-price priority also gives proposers some influence and doesn't guarantee fairness by arrival time.

2.5. Consensus phases

Agreement proceeds through prepare, pre-commit and commit phases, using certificates and persistent voting locks. HotStuff [6] is a reference design, although it isn't a proof of this composition of group and global consensus.

Prepare: a validator checks the proposal's parent, epoch, slot and resource bounds, then re-executes its transactions, and it only votes if the resulting state root, receipts root and gas used match the proposal. It stores the proposal body before voting, so a prepare certificate proves that a quorum holds the data. **Pre-commit:** validators verify the prepare certificate and their existing locks before voting. **Commit:** a block is final once its commit certificate forms.

Because a validator records every vote durably before signing it, a crash and restart cannot make it contradict itself. If for whatever reason a leader doesn't produce its block, the round times out and the next proposer takes over with existing locks still held. There can be many reasons for this, such as the validator being down for maintenance, under attack or choosing to omit a block for malicious or moral reasons. In any of these cases the validator isn't slashed and just misses out on the slot reward.

The next batch can be prepared while the current one is still completing, although dependent work stays speculative until its predecessor is final. Nodes monitor clock skew to support the 500 ms slot target, but it is quorum evidence and locks that decide agreement and the clocks have no say in it.

2.6. Staking and slashing

Validators register an ML-DSA key with a proof of possession and a reveal root (Section 2.2), and accept delegated JOLT through share-based stake positions. Withdrawals pass through an unbonding period, which is how stake stays accountable for recent behaviour. A missed slot costs only that slot's reward, while signing two conflicting votes for the same height, round and phase is slashable. Anyone can submit the two votes as evidence, and the validator then loses part of its stake and may be barred from validating in the future. The reporter receives a share of the penalty and the remainder becomes protocol revenue (Section 3.9). Delegators share any loss in proportion to their stake, and this includes stake that is still unbonding.

2.7. Network topology and data availability

Dense links within small groups reduce forwarding hops. Between groups Joltzu uses a layered ring mesh topology with randomized cross-group links and gossip. Redundant routes improve resilience, but they do not replace availability checks. Validators authenticate one another with mutual TLS and apply per-peer quotas, so a flooding peer only loses its own traffic.

Storing proposal bodies before prepare votes gives a simple availability guarantee, because whatever is certified is held by a quorum. Erasure-coded dissemination with Reed-Solomon-style fragments would use less bandwidth than full copies, but it can only replace them if its reconstruction thresholds, commitments and recovery from data withholding preserve that guarantee.

2.8. Formal verification

We specify the group and global protocols in TLA+ [7] and prove their safety in Appendix A. The ledger never finalizes two blocks at the same height, every included batch is valid and was finalized by its own group, and every finalized block is stored by an honest validator. These properties hold with weighted quorums and locks, under arbitrary message delays, equivocating leaders and Byzantine validators that sign anything. The TLC model checker confirms them exhaustively on bounded configurations, including one in which a whole group is Byzantine, and that group can equivocate but cannot fork the ledger or insert an invalid batch. These results don't cover liveness, epoch transitions or RANDJAO.

3. Execution and Accounts

The EVM is the only contract environment on Joltzu. Contracts are ordinary EVM bytecode which validators execute natively on their host hardware, they share one state with the native services and they call those services through precompiles at fixed addresses. Where Joltzu differs from Ethereum is the account, because validators enforce each account's policy on every execution path (Figure 4).

3.1. EVM semantics and application security

Within a published compatibility scope, contracts keep the EVM's arithmetic, storage, calls, callbacks, reverts and gas metering [4], and addresses are still 20 bytes. Joltzu does add rules of its own, such as network eligibility, asset policy, outflow limits and holds, but we state these as explicit additions

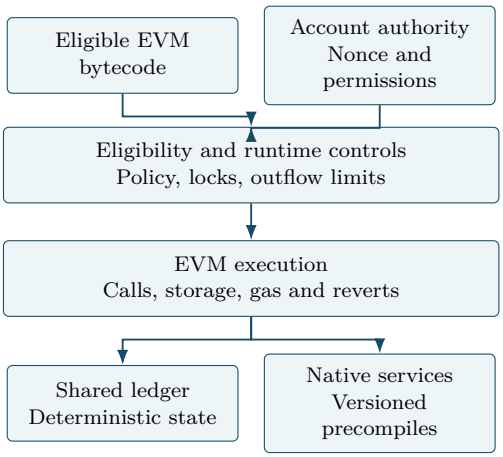


Figure 4: *EVM contracts use shared state and native services, subject to account authorization and the protocol's safety controls. Optimized execution has to keep the specified EVM behaviour and follow Joltzu's execution policies.*

and do not silently change how the EVM behaves. The protocol does not ban reentrancy, so it is up to developers to protect their application invariants with reentrancy guards, the checks-effects-interactions pattern and testing [8]. Screening (Section 5.1) looks at these protections, including the ones against cross-contract and read-only reentrancy.

3.2. Native smart accounts

Every address has an account policy, which sets who can act for it and within what limits. A new key-derived address starts with an implicit policy whose only signer is its own key, which is how it behaves like an ordinary Ethereum account and needs no wallet contract. Later on the owner can enable multisig, guardian recovery and bounded grants at the same address, and the account keeps all of its balances and positions. Contracts check a signature from any account through the standard ERC-1271 interface [9], and the protocol answers from that account's policy, implicit or explicit, so an address with no code answers too and a standard wallet's signature works unchanged. The `ecrecover` precompile is deprecated. It stays for compatibility with existing contracts, but it names a key rather than an account, so it speaks only for a key that can still act alone and cannot represent a threshold or post-quantum signer.

Because validators are the ones enforcing the active policy, it doesn't matter which wallet, relayer or keeper submits a transaction, and the policy still applies when the wallet app is closed. Joltzu Wallet (Section 8) is the main interface to these accounts.

3.3. Multisig and payment thresholds

An account can require m of n approvals, with any mix of ordinary keys and post-quantum ML-DSA keys, and there is no separate contract to deploy [10]. Each signer counts once. An approval is tied to the account, chain, policy version, nonce, complete action and expiry. This means a relayer cannot alter an approved action or reserve funds merely by collecting signatures. Whoever submits an approval pays its gas, which might be one of the signers' own addresses, a co-signer or an exchange's relayer, and the approval can also name a relay fee, in JOLT or a listed stablecoin such as USDC, that the account pays the submitter before the plan runs, together with the least gas the plan has to receive. The fee stays paid if the plan fails, so a relayer can rely on it, and because the minimum gas is signed a relayer cannot collect the fee by starving the plan. One of the signers can be JSN's co-signer, which only signs requests that meet rules the owner has set, such as an IP allowlist or an authenticator code (Section 13.5).

For routine use an account can set a lower payment threshold. For example, a three-signer account might let any one signer pay up to a fixed JOLT amount to a plain address, or perform a short list of capped native actions such as placing a bid or repaying a loan. Contract calls, token approvals and policy changes still need the full administrative threshold. If signers go missing neither threshold gets any lower, and changing the policy invalidates outstanding approvals.

3.4. Recovery

An owner who enables guardian recovery sets a guardian threshold and a delay. All the guardians can do is propose a replacement signer set, which the owner can cancel during the delay, and they do not gain any spending authority. Recovery keeps the account's locks, debts, holds and funded commitments.

3.5. Safeguards

An account can also opt into safeguards, which slow down whoever controls it, so that a stolen key, a compromised signing service or a quorum of deceived signers has to wait in public before it can move much value or change who controls the account. The owner sets a delay of between one hour and 30 days, a daily allowance for each asset they want to limit and up to four cancellers, which might be the owner's own offline key or a multisig of their guardians. Every debit of a limited asset draws on its allowance whichever path it takes, including the account's own key, plans, payment thresholds, grants and contract allowances, and the allowance refills evenly over a day, so splitting a withdrawal into smaller ones doesn't help. Anything larger has to be scheduled. The signers approve the plan in a separate signing domain, which means an approval to run a plan now can't be queued and a queued one can't be run now, and once the delay has passed anyone can release the plan within the following week. Until then the account or any of its cancellers can cancel it, and a policy change that takes effect first makes it lapse.

Safeguards also delay any change that loosens the account's authority. A new signer set or threshold, a new payment threshold or a looser safeguard waits for the delay in the same way, while a change that only tightens the safeguards applies at once, so an attacker who tricks the signers

into replacing them ends up with a public, cancellable pending change instead of the account. Every scheduled plan and change emits an event, which lets wallets and watchers alert the owner while there is still time to cancel.

Safeguards cannot tell the owner apart from an attacker who also holds the whole threshold. Such an attacker can still take the daily allowance and cancel the owner's own changes, while the cancellers stop anything larger, and that buys time for bonded pauses and holds on the rest (Section 5.7). A canceller can only cancel, but it can cancel any waiting change, including the one that would remove it, since otherwise an attacker could first schedule the removal of every canceller, and so a canceller should be someone the owner would trust with a veto.

3.6. Grants and delegated authority

Because approving every action reduces the UX, an owner can issue bounded grants instead. A payment grant lets a session key pay JOLT to a fixed recipient within per-payment, daily and lifetime limits, an operation grant permits one exact contract call a limited number of times and is pinned to the target's code, and a token grant permits bounded token transfers. Every grant expires. Joltzu Wallet builds its website-bound permissions on these primitives (Section 8.4).

We deliberately treat grants and funded commitments differently. A grant or recurring mandate doesn't reserve any funds, so a policy change, for instance after a lost device, revokes it. A funded commitment, such as a voucher budget, escrow or bequest, has already moved its funds out of the spendable balance, which is why it survives policy changes and recovery and only ends under its own terms (Section 9.5).

3.7. Native services and precompiles

Joltzu exposes its native services, which include markets, lending, payments, oracles and staking, through versioned precompiles at fixed addresses. Each precompile defines its inputs, outputs, permissions, errors and gas costs, which gives developers a clear interface to call. A precompile call doesn't get around the account rules, because it carries the authenticated account context and obeys the caller's policy, locks and outflow limits. Native state changes are journaled with EVM state, which is how the two commit or roll back together within a transaction.

3.8. Native JOLT and its ERC-20 interface

JOLT is the chain's native asset, and it is used both to pay for activity and to give participants something at stake. It pays transaction and native-service fees, and transaction fees go to the block proposer by default. JOLT also backs AI-worker bonds and retention-lease collateral (Sections 6 and 4), and validators and delegators stake it (Section 2.6). JOLT's supply, allocation and distribution are outside the scope of this paper.

JOLT is also the EVM's value balance. A built-in interface exposes it through the ERC-20 functions `balanceOf`, `transfer`, `approve` and `transferFrom`, so there is no wrapping [11]. Native transfers and ERC-20 calls read and update the same balance under the same safety checks, and a token allowance grants permission without creating a second balance (Figure 5). Canonical transfer events stop indexers from counting a movement twice. There is no wrapped Jolt token which improves UX.

3.9. Protocol revenue and the treasury

Some JOLT is taken from participants as a penalty or a fee rather than paid for execution: the part of a slash left after the reporter's share (Section 2.6), the bond of a pause the Council rejects (Section 5.7) and alias registration fees (Section 14.6). Joltzu doesn't burn all of it. In our illustrative parameters half is burned, a quarter goes to the reward reserve of the JOLT/USDC farm (Section 12.3) and a quarter goes to the treasury. Burning offsets new issuance, the farm's share ties liquidity rewards to how much the network is used, and the treasury pays for audits, development and other work the community chooses. The JAO governs the treasury (Section 16.1), not the Security Council, so a Council that rejects a pause cannot direct the bond it slashes. Transaction fees are separate and go to the block proposer.

4. State, Storage and Node Architecture

Validators run the C++ reference node, while JSN's off-chain services run outside it (Section 13). This section covers state, storage, validator structure and performance work. One million transactions per second is a throughput target. This is phenomenally hard, and what is achievable depends on parallelism, state contention, signature checks, durable writes and network bandwidth, which is why every optimization has to be justified by workload measurements.

4.1. Authenticated state and history

EVM contracts and native services share one Merkle-Patricia state trie. Each block identifier commits to the block's contents, state root, receipts root and an index of its logs, so any account, storage value or event can be proved against a finalized block. A native precompile keeps the state roots and block hashes of recent blocks, and a permanent accumulator lets a short Merkle branch authenticate the roots of any older block. This means contracts can verify historical state and event proofs on-chain without trusting an archive operator.

4.2. Light clients

A light client starts from a trusted committee and checkpoint, and checks each block's commit certificate, parent link, timestamps and randomness chain. It tracks committee changes through certified history, so trusting the first committee is enough to follow later epochs. Instead of re-executing transactions, it requests state proofs for the data

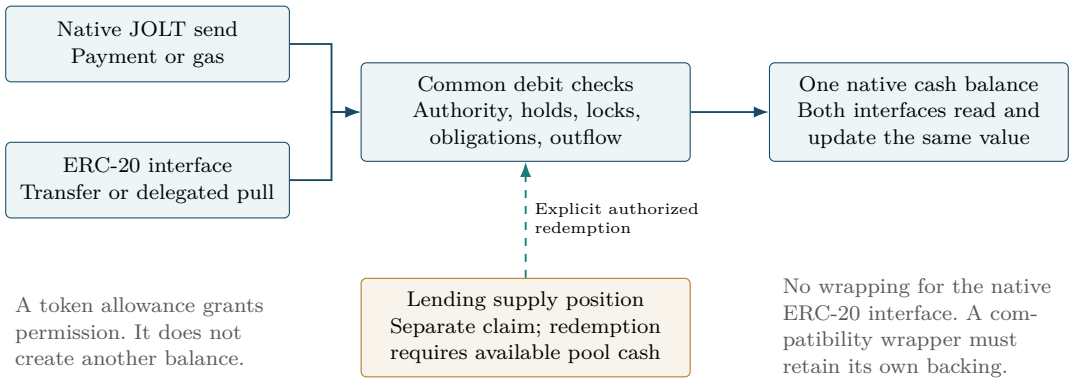


Figure 5: *Native and ERC-20 operations use the same JOLT cash balance and the same safety checks. Lending claims and wrapper backing are accounted for separately, and neither creates additional assets.*

it needs. How recent its starting checkpoint has to be is a question of weak subjectivity (Section 17.3).

4.3. History retention and filter nodes

Validators need recent state to execute, but not all of history. Archive peers serve certified history and proofs to clients and to nodes that are catching up, and retention leases fund that service. An archive holder posts collateral and a named client prepays the price, and anyone may then challenge a covered state, log or receipt record for a fee. Answers are verified on-chain against committed roots. If a challenge is left unanswered for 256 blocks (an illustrative value) the lease fails, and the holder's remaining collateral and any unearned prepayment go to the client. Griefing is limited because each lease accepts a bounded number of pending challenges.

We propose filter nodes for projects which only process certain events, because archive nodes can be expensive to run. A filter node retains selected ERC-20, ERC-721 and ERC-1155 transfer and approval streams to make indexing cheaper. It answers only the queries it supports, so other history still requires an archive, and resynchronizing cannot leave gaps in these streams.

4.4. Storage

The storage layer has to keep disk work cheap and stay consistent when something fails. Joltzu uses RocksDB for the main log-structured store and write-ahead log [12]. Caches, Bloom filters, batched writes and compaction reduce read, write and space amplification. For consistency, execution state and consensus safety records live in separate stores, each store commits atomically, and records that need to change together share a commit boundary. A validator may spread storage across nearby machines, but reads and durable commits still need one consistent failure boundary, and spreading storage this way doesn't reduce replication across validators.

4.5. Validator process architecture

A validator is one logical node built from cooperating components. Networking, signature verification, execution and storage can be spread across cores and nearby machines, but they are still one validator with one consistent durable state. That is why benchmarks need to include every machine, the cost of coordinating them and their failure boundary.

A validator's ML-DSA signing key lives in a separate signer process. The node only requests a signature for a vote after recording it durably, and it verifies the returned signature before using it.

4.6. Performance engineering

Instruction dispatch and native compilation

Using an interpreter is not optimal, because it spends time on dispatch and stack management as well as the computation itself, so Joltzu compiles contract bytecode to machine code and executes it natively on the host hardware. For instance, Listing 1 shows this overhead for a simple addition, while a translated representation can express it directly.

```
switch (opcode) {
  case add: {
    const auto s = stack.pop();
    stack.top() = stack.top() + s;
    break;
  }
}
```

Listing 1: *Illustrative interpreter dispatch for addition.*

```
result = a + b;
```

Listing 2: *Conceptual direct representation of addition.*

Removing dispatch and stack traffic saves work, but the translated code still has to preserve correct 256-bit arithmetic, operand validation, gas accounting and exceptions, and host integer types cannot silently replace EVM words.

Interpreted, compiled and native execution have to produce identical results, state changes, failures and gas charges on all supported hardware, checked against a reference interpreter.

Compile-time constants

C++ compilers are very good at optimizing at compile time, and specialization can remove runtime branches for fixed configuration. This means mainnet and testnet are completely separate builds, each with an explicit network identity.

Branching and dispatch

Branches like the one in Listing 3 incur a cost with the CPU's branch predictor, and unpredictable ones can disrupt the pipeline. Branchless alternatives will make frequently executed paths faster, especially when branches are hit close to 50-50.

```
if (condition) {
  // Perform the first operation.
} else {
  // Perform the alternative operation.
}
```

Listing 3: *Conditional control flow in C++.*

Virtual dispatch is a similar trade-off. Joltzu uses direct calls where the target is known, which removes repeated vtable lookups from hot paths without imposing one programming style on the whole node.

Memory and hardware acceleration

Over time many small dynamically allocated objects leave small pockets on the heap which cannot be used for anything bigger. Joltzu reduces this fragmentation, and allocator overhead, with memory pools which recycle short-lived transaction and message allocations, and explicit ownership and lifetimes keep this reuse safe across threads.

Batched signature verification can run on CPUs or accelerators, and SYCL is used for shared C++ compute code on the GPU with a CPU fallback. The gains depend on hardware support, transfer overhead and batch size, and every path has to produce identical protocol results.

Network encoding and C++ layout

A typical ABI may add padding to a structure such as the following:

```
#include <cstdint>

struct SomeStruct {
  std::uint8_t x;  // Often followed by 3 pad bytes.
  std::uint32_t y;
};
```

Listing 4: *Illustrative aligned structure; layout is ABI-dependent.*

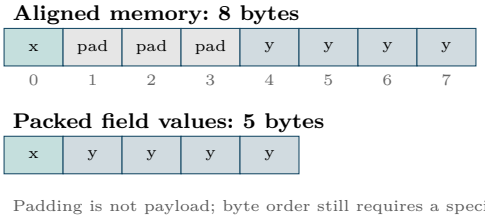


Figure 6: *Aligned and packed layouts for Listing 4, assuming four-byte alignment for* `uint32_t`*. A portable wire format also has to define byte order and field access.*

With four-byte alignment for `y`, this structure commonly occupies eight bytes, although its field values need only five packed bytes (Figure 6). Unpacking data off the wire takes time, so Joltzu's network format lets messages be read in place, without a separate allocation and unpacking step for every field. Cap'n Proto is a reference for an encoding with a defined, word-oriented layout [13]. Because portability depends on fixed widths, byte order, alignment, bounds checks and versioning, compiler-specific layouts and uninitialized padding stay out of the wire format.

4.7. Congestion controls

Since one busy application should not be able to crowd out the others, Joltzu gives each contract and service its own deterministic budget [14]. A call that touches several services draws on each relevant budget within total replicated capacity, and the budgets have to account for address splitting, fair admission, maintenance traffic and the expiry of deferred work.

5. Safety Model

Everyday users cannot audit each application before using it, and even careful users occasionally sign something malicious, such as an approval for a contract that later steals their funds. So Joltzu layers its defences. Wallet screening acts before a transaction is signed, then network eligibility, account policy and safeguards, contract invariants and outflow limits act while it executes, and finally monitoring, bonded pauses, holds and the Security Council act once something is detected (Figure 7).

5.1. Wallet screening and network eligibility

Joltzu distinguishes between *reviewed* code, which has verified source and security screening, and *admitted* code and assets, which network policy permits. Verification matches the source and compiler settings to the bytecode. Screening looks at proxy upgrades, factories, dependencies, privileges and initialization, and it records the tools, tests, AI findings and implementation that were reviewed. By default Joltzu Wallet requires verified source and a security scan before an action, although the user can override that requirement for one approved action.

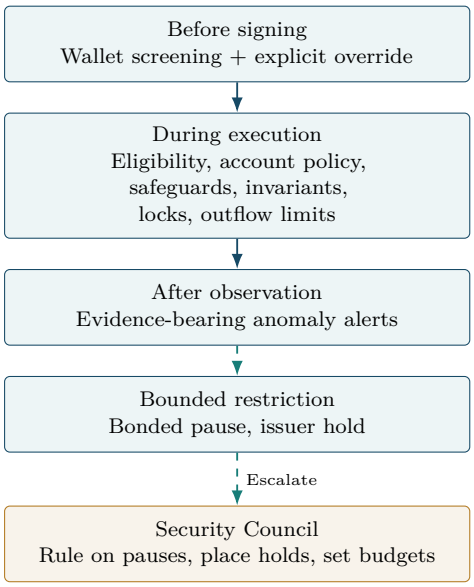


Figure 7: *Defences act before signing, during execution and after detection. A pause or a hold restricts a specific amount, and the Council rules on pauses, places holds and adjusts outflow budgets without halting the network.*

Network eligibility is different because it's a consensus rule that applies to every submitter, whichever wallet they use. It covers asset admission, per-asset holder permissions and, optionally, a set of admitted runtime code hashes. When code eligibility is enforced, every contract creation has to deploy admitted code, and that includes contracts created by a factory, so a proxy can only delegate to admitted implementations. A code-admission committee admits or revokes hashes for future deployments, and revoking a hash leaves code that is already deployed as it is. If a user overrides screening, that only waives the wallet check, while eligibility, account permissions, locks and incident restrictions all still apply. Neither review nor admission guarantees that code is free of defects.

5.2. Circuit breakers

A circuit breaker limits how quickly value can leave, so that a theft which gets past every other check runs into a ceiling before it empties its target. Joltzu's circuit breakers are outflow budgets, which apply to an asset, protocol, wallet or service. Each budget has a capacity, a refill rate and a maximum single transfer, and it works like a bucket that slowly refills up to its capacity. For instance, an asset with a capacity of 100 million, a refill of 1 million an hour and a maximum transfer of 25 million lets an attacker take at most 100 million straight away, across at least four transfers, and after that only 24 million a day (these numbers are illustrative). That is the difference between losing everything in two transactions, as Ronin did, and having hours or days to respond.

A budget counts every actual debit of the asset whichever path it takes, including EVM value movements, native transfers, market settlement and firm quotes. Splitting a withdrawal across transactions or accounts doesn't help because all the parts draw on the same budget, and because budgets are measured in asset units and protocol time, stale or manipulated prices cannot increase capacity. A transfer that exceeds the available budget fails as a whole, and partial settlement is only possible through an explicit partial-fill interface. Capacity shares a journal with balances, so a failed settlement or a revert restores both. A resting order isn't resized when someone else uses the budget, and its settlement simply has to pass the same check. When capacity recovers, queued actions are rechecked so that a backlog does not drain the restored budget all at once.

A circuit breaker doesn't stop the network. Protocol fees sit outside the budgets, so blocks keep being produced and operations that move no value carry on. The Security Council can set a temporary budget during an incident with more than two-thirds of its whole membership (Section 5.6), and that budget has to expire, within a day in our illustrative parameters, after which it imposes no restriction unless the Council publicly renews it. Updating an active budget keeps the capacity that remains, clamped to the new limit, and a renewed budget starts empty, so neither action can be used to quietly refill it.

The trade-off is that a budget is shared. Once an attacker has used it up, honest users of that asset wait for the refill too, and legitimate large flows need arranging in advance (Section 5.6). We accept this because a delayed withdrawal can be put right and a stolen one usually cannot, but it does mean a budget needs setting high enough that normal activity never reaches it.

Targeted JOLT restrictions

Emergency caps can apply to direct sends, internal calls, markets, lending, liquidations and wrappers while block processing carries on. Native JOLT and its representations share one exposure account. Cancellations, accounting and oracle updates can still go through, and any recovery exception binds its purpose, recipient, amount and expiry.

5.3. Contract invariants

Outflow budgets limit what an exploit takes once it is under way, but many application exploits end in a state that the protocol's own authors would have called impossible, such as debt that its collateral no longer covers or a pool that holds less than its shares are owed. A contract can therefore name an invariant checker, which is a separate contract with a read-only function that says whether the protected contract is still sound. After every transaction that ran the contract's code or debited it, validators call the checker with a bounded share of the transaction's own gas, and if the checker doesn't confirm that the invariant holds, the whole transaction reverts. The check runs once at the end of the transaction rather than after each call, so flash loans and other temporary imbalances still work as long as the contract ends up

sound, and because the call is read-only it cannot change state or move value. Only the contract itself can name or clear its checker, which means its code has to provide for that, and a contract that can clear its checker should only do so through its own timelock.

An invariant only enforces what its code states, so a wrong or incomplete one lets an exploit through and one that is too strict reverts honest transactions. It also only sees state on Joltzu, which means it cannot catch a bridge accepting a forged message about another chain. Its value is that it doesn't depend on anyone finding the bug, so an invariant written when a protocol is generated (Section 14.1) still protects it years later, when a new class of bug turns up in its code or its compiler.

5.4. Voluntary time locks

An owner can lock selected assets until a chosen date, up to a year ahead. A lock can be extended but never shortened, and nobody can release it early, whether that is the owner, a delegate, a guardian or the Council. Locked funds count as a hard obligation, so conflicting allowances, debts, margin and funded commitments have to be settled first. Incoming funds do not extend an existing lock. Recovery or inheritance can establish a future controller, but neither of them can release the lock early.

5.5. Issuer controls for approved assets

Each approved asset other than JOLT has an issuer committee, which can place an expiring, amount-specific hold on a holder's balance with a recorded incident reference. It can also set holder permissions, such as whether an account may send, receive or trade the asset, and it can delist and relist the asset. Delisting closes the asset's markets to new exposure, including AMM pools and their farms (Section 12.3), orders, quotes and new loans, while existing balances stay transferable and positions can still be exited. A hold only stops funds from moving. It cannot move, mint or burn them, and an issuer committee has no say over JOLT or any other asset.

5.6. The Security Council and who can do what

The validator Security Council coordinates incident response. It sets temporary outflow budgets for approved assets, including JOLT, it places and releases expiring holds on specific funds in any account or contract, it rules on bonded pauses (Section 5.7) and it approves ERC-20 assets for listing (Section 12.1). Each action binds a nonce and an expiry, is recorded publicly and needs signatures from more than two-thirds of the whole Council, counting absent members. It cannot pause the network or release voluntary locks, and it cannot move, mint or burn user funds. Recovery changes the state from that point on, and finalized history is never reversed.

A hold is how the Council freezes stolen funds without stopping anyone else. It names one asset, one account or contract, an amount and an incident, and it lasts at most 30 days in our illustrative parameters, after which it ends unless the Council publicly places it again. The held amount cannot be spent, pledged or used to pay fees, while the rest of the account carries on as normal and the funds still belong to whoever held them. Native service accounts and native pools cannot be held, so that markets keep settling, and outflow budgets protect those instead. We know this is a real power, because a compromised two-thirds of the Council could freeze any account for as long as it kept renewing the hold and a Council that can freeze funds will be asked to by courts and regulators. We accept that because the alternative is what other networks have done during exploits, which is to improvise powers that their users were never told about (Section 7), and so we state this one up front and put every use of it on the public record.

The other powers go to narrower bodies (Table 1). Remediation that needs new code follows the code-admission path, and estates follow the owner's own verifiers. Because authority is divided this way, a compromised committee can only do what its own powers allow.

5.7. Bonded pauses

Collecting signatures from more than two-thirds of the Council can take hours and stolen funds do not wait, so anyone who sees an exploit can stop the funds themselves by putting up collateral. A reporter names an asset, an account or contract, an amount and a commitment to their evidence, and posts a JOLT bond that grows with the amount, at a rate set for each eligible asset, and validators apply the pause as soon as that transaction executes, without anybody's signature. Since the protocol cannot judge the evidence, the pause is short, lasting six hours in our illustrative parameters, and it only stops that amount from moving, so nothing is moved, minted or burned. A pause asking for more than the target's unlocked balance is reduced to fit, and its bond with it, which means a target cannot dodge a pause by sending a few units away first. Nobody except the Council can lift a pause early, including the reporter who placed it.

The Council then has three days to rule. If it upholds the pause the bond is returned, the pause becomes a Council hold with no gap in between, and the reporter earns a bounty of up to ten times the bond, which pays out after 30 days so that the Council can cancel it if the reporter turns out to be tied to the attacker. If the Council finds the pause unjustified the funds are released at once, the bond is slashed as protocol revenue (Section 3.9), and the Council can refuse new pauses on that target for up to 30 days, so that one account cannot be frozen over and over. If the Council doesn't rule in time the bond goes back with no bounty, because a reporter should not pay for an absent Council.

Bounties come from a dedicated reserve which anyone can fund, including protocols that want to reward people for watching them. Slashed bonds never go to that reserve, so a

Authority	Can	Cannot
Account owner	Spend, issue grants, lock funds, set up recovery, safeguards and bequests	Shorten a lock, skip their own safeguard delay or bypass holds and outflow limits
Guardians	Propose a replacement signer set after a delay	Spend funds, or proceed if the owner cancels
Cancellers	Cancel an account's scheduled plans and waiting changes	Spend funds or make any change themselves
Security Council	Set temporary outflow budgets; place and release holds; rule on bonded pauses; approve asset listings	Move, mint or burn funds; pause the network; release voluntary locks
Reporters and pools	Pause a specific amount for a few hours by posting a bond	Extend or lift a pause; pause native pools; recover a slashed bond
Issuer committee	Hold amounts, set holder permissions, delist or relist its own asset	Act on JOLT or other assets; move funds
Code-admission committee	Admit or revoke runtime code for future deployments	Change deployed code or existing state
AI agents	Report evidence; post bonded pauses like any other reporter	Transfer assets, edit code or stop block processing
Bequest verifiers	Start a bequest's notice period with a death report	Redirect, shorten or settle a bequest

Table 1: *We divide authority so that each party holds only narrow powers. Every committee acts only with signatures from more than two-thirds of its full membership, and each action expires and cannot be replayed.*

compromised Council cannot slash one reporter's bond and then pay it to itself as a bounty.

Collateral can be pooled, for those who want to back a good watcher without running one. A pool takes JOLT deposits in exchange for shares, and its operator names the reporters who can draw on it, each with an allowance that shrinks for good when one of their bonds is slashed, so one careless reporter cannot lose everybody's deposit. Returned bonds and bounties go back to the depositors, less a share for the reporter which is fixed when the pool is created, and depositors bear any slashing, which is why withdrawals wait until every open bond of the pool has been ruled on. Addresses on the Council's roster cannot report or deposit, although we cannot stop a member from using another address, and that is one more reason why bounties are delayed and cancellable.

A pause can also be used as a weapon. Someone could pause a lending contract so that its liquidations fail and earn more from the resulting bad debt than the bond costs them, and a larger bond only makes that more expensive, which is why native pools cannot be paused and why the Council can release a pause as soon as it sees one. An attacker could also try to crowd out a real pause with small ones of their own, so when an account's pause slots are full a larger pause displaces the smallest, whose bond stays at stake until the ruling. Pauses and holds both only catch funds that are still there, which means neither can stop a drain inside a single transaction or recover funds that have already left the chain.

Pre-arranged outflows

Legitimate large outflows can be arranged in advance through a Council-approved allowance, which binds the account, asset, destination, total amount, window and nonce. It gives no access to user keys and does not waive KYC, locks or active holds.

Bridges

Because bridging is one of the ways stolen funds get dispersed, bridge flows count towards per-account and aggregate outflow limits. To keep track of bridges the Council maintains a registry of bridge contracts, which records each bridge's source and destination networks, supported assets, implementations and accounting adapters. Registering a bridge does not vouch for its external security. The Council can also refresh a named account's bridge allowance, and each refresh records the asset, amount, reason, window and a single-use identifier. These refreshes accumulate visibly so that they do not silently reset budgets, and they never release holds or voluntary locks. We will specify the registry together with the first supported bridge.

6. AI Agent Network

The AI Agent Network screens contracts, detects threats and helps users with their requests, using bonded workers that run models off-chain as part of JSN (Section 13), outside consensus, and return signed results. The protocol records each request, keeps its fee in escrow, verifies the signed results and enforces accepted holds, but it does not require validators to agree on an open-ended natural-language answer. Joltzu keeps inference outside block execution so that no ordinary transaction waits for a model and validators need no AI hardware.

Reproducing a model's output takes more than a shared prompt and random seed, because hardware, quantization, kernels and runtime versions can all change the result [15]. So each task names an execution profile for a selected worker

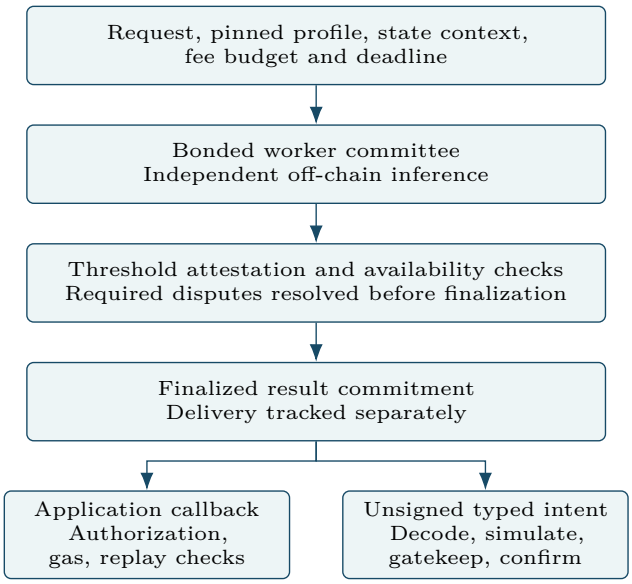


Figure 8: *A committee of workers produces an accepted result, and whoever consumes it still checks that it has the authority to act. Security alerts use bounded quarantine, with Council intervention authorized separately.*

committee, and a rotating coordinator collects the responses (Figure 8).

6.1. Requests and committees

A request fixes its inputs, execution profile, output schema, state snapshot, deadline, callback and fee cap. Protocol randomness then selects workers with the required profile, and each of them runs the task independently and signs a result bound to that request, with the signature binding the chain, service, nonce, committee, model, inputs, output and delivery limits. The weighted signing threshold counts each signer once and stays the same when workers are absent [16]. At any point a request is pending, proposed, challenged, finalized, expired or cancelled, with challenges resolved before it finalizes and ledger order settling any race with a cancellation. Callbacks are tracked separately and get retried within their budget under single-use identifiers.

An execution profile pins the model weights, tokenizer, prompt, formats, decoding settings, seed, arithmetic, runtime and hardware. Deterministic tasks require reproducible outputs, while for subjective tasks the workers evaluate one normalized candidate against a rubric. Votes for different outputs cannot form a quorum, and workers agreeing is no proof that the answer is factually true. Optimistic replay [17] and proofs for small quantized models are still research, since both need executable verifiers, available data and bounded challenge costs, and proving a computation doesn't prove what its output means.

6.2. Agent identities and work records

A versioned registry, drawing on ERC-8004 discovery patterns [18], records operators, keys, profiles, endpoints and fees, and receipts cover requests, results, delivery and upheld disputes. If an agent changes operator its history stays with it, but its reputation is not carried over. Bad actors could try to build up reputation with self-generated jobs and coordinated ratings, so worker selection has to resist both. A good reputation doesn't prove that an agent is right and it doesn't give the agent any authority. There is no agent token either.

6.3. Security agents and quarantine

Security agents inspect approvals, privileges, oracle deviations and fund flows, and they begin in shadow mode, where they observe without acting, so that we can measure false alarms, missed incidents, latency and drift under evasion attempts and poisoned telemetry [19]. Independent attestations require independent operators. Like the rest of JSN, the agent workers are distributed, not decentralized, because one operator runs all of them (Section 13).

We phase in enforcement: monitoring first, then recommendations to the Council, then bonded pauses posted by the agents themselves (Section 5.7). An agent has no special standing, so its pauses need the same bond, last the same few hours and face the same ruling as anyone else's, and a model that raises false alarms loses its operator money. Once a pause is accepted, validators enforce it deterministically, without consulting a model for each transaction. Agents inspect finalized blocks, which means they can only restrict later movements. They cannot transfer assets, edit code or stop block processing.

6.4. Incident tracing

Tracing follows an incident's funds across accounts. Protocols can register a flow profile of the deposits, withdrawals, counterparties and net outflows they expect, and weakening a registered profile requires its stated delay and authority. An accepted incident record identifies the source transactions, evidence, asset, attributed amount and review status. The candidate tracing rule divides mixed transfers of the same asset proportionally, with fixed rounding, so attribution is conserved from hop to hop and cannot multiply across branches. Holds only cover attributed value that is still present. For instance, a 40-JOLT hold on a 100-JOLT account leaves 60 JOLT under ordinary rules. Swaps, pools and bridges need explicit adapters, and tracing has no way to freeze every pool participant or to guarantee off-chain recovery. The wallet shows the restricted amounts, reasons and review status.

6.5. Natural-language intents

The agent turns a request into inspectable calls and limits, and account authorization always stays outside the model.

Example: prepare a swap

> *Swap up to 30 USDC for at least 100 JOLT to my account using approved assets. Expire ten minutes after signing. Show the route, approvals and total fee limit before asking me to confirm.*

The amounts are illustrative. The wallet resolves the addresses and deadlines, decodes the actual calldata and simulates it against current state. A signed intent binds the chain, target, arguments, input cap, minimum output, recipient, nonce, expiry and gas budget. If a route is unavailable those limits still stand. Workers don't receive user keys at all. SDK helpers such as `agent.orderBook(prompt)` and `agent.estimate(prompt)` provide versioned interfaces for submitting and cancelling requests. RPC previews are advisory, and contracts only consume paid, finalized results.

A price-triggered intent fixes the oracle feed, threshold, asset, order limits, expiry and whether it recurs. An executor watches finalized oracle reports and submits the call, and execution then rechecks the report and the signed limits, which means a stale or disputed report can't trigger it.

Comments, websites, metadata and tool outputs are untrusted data, which is why they can't change enforced permissions or a scanner's authority. Workers generally see their inputs, so confidential inference needs its own rules for custody, encryption and retention. Private keys and recovery phrases must never go in a request, and hashing predictable private data doesn't hide it.

6.6. Fees and upgrades

A request's capped JOLT escrow pays for inference, replication, availability, challenges and callback gas, and any unused funds are returned. Bonds penalize provable faults and leave subjective disagreement alone. Model updates come with published evaluations and activation epochs, while existing jobs keep their original profile.

7. Lessons from Major Exploits

More than $3.4 billion was stolen from crypto services and users in 2025, and $1.5 billion of that came from Bybit alone [20]. TRM Labs attributes about three quarters of 2025 losses to compromised keys, signing flows and infrastructure, while code exploits were more numerous but smaller [21], and the pattern continued in the first half of 2026 [22]. Then on 29 October 2026 an attacker took about $6.4 billion from Binance, which is more than four times Bybit's loss and the largest theft on record (Section 7.1) [23].

Table 2 lists the largest incidents on the rekt.news leaderboard [24], plus the Binance theft, which the leaderboard had not yet listed at the time of writing. For each one the table shows, counterfactually, the Joltzu controls that would have applied. We left out frauds and failures of key generation that no protocol can do anything about. Joltzu cannot stop keys being stolen or bugs being written either. What it changes is what a stolen key can sign, how much value can leave and how quickly, whether a transaction that breaks a protocol's own accounting can complete at all and how fast stolen funds can move, and this means owners, monitoring and the Council get more time to respond.

7.1. The Binance theft

On 29 October 2026 an attacker drained about $6.4 billion from Binance hot wallets across about ten blockchains, most of it within twenty minutes [23, 46]. Binance's initial investigation points to a compromised DevOps engineer's laptop, which let the attacker reach a service container and submit withdrawals, and the investigation found no evidence that a wallet key had leaked [1]. Binance's own systems appear to have signed the transfers. On Ethereum, 50 pre-signed EIP-7702 authorizations turned one hot wallet into a sweeper that the attacker controlled, and it kept capturing new deposits for about three and a half hours. Before issuers could act, the attacker swapped about $786 million of stablecoins for ETH and TRX and split 18,270 BTC across 19 addresses. Stablecoin issuers and others froze about $2.8 billion, 94% of it in stablecoins, but the ETH and bitcoin remain beyond any issuer's reach. TRM Labs attributes the theft with high confidence to North Korea's TraderTraitor, although no government has confirmed this. Binance, which suspended withdrawals and says it will cover all losses, has not yet published a root cause.

On Joltzu, protecting the signing service would still be the exchange's job, but the theft would have unfolded differently:

- Outflow budgets on exchange hot wallets would have capped what any signer could move in each period, and a service that suddenly sent billions would have hit its budget and raised an alarm within minutes (Section 5.2).
- With safeguards on each hot wallet, a withdrawal service that signed whatever it was given could only have sent the daily allowance, and every larger withdrawal would have waited, in public, for a delay in which the exchange's security team, as a canceller, could have cancelled it (Section 3.5).
- Joltzu rejects EIP-7702 transactions. The only way a native account changes how it authorizes spending is through an administrative policy change. With an administrative threshold above the payment threshold, a compromised payment service would not have been able to turn a wallet into a standing sweeper (Section 3.3).
- Anyone watching could have paused whatever was still in the attacker's accounts by posting a bond, JOLT included, and the Council could have turned those pauses into holds (Section 5.7).
- Issuers of approved assets could have placed recorded, bounded holds on the attributed funds, as Tether and Circle did here, without halting the network (Section 5.5).

Incident	Date	Loss	What went wrong	Joltzu
Binance	Oct 2026	$6.4B	A compromised engineer's laptop reached the withdrawal service; Binance's systems signed the transfers, one hot wallet via an EIP-7702 sweeper [1, 23]	Delayed, capped, held
Bybit	Feb 2025	$1.46B	Tampered wallet interface; signers blind-signed a call that replaced the multisig's implementation [2]	Blocked, delayed, held
Ronin bridge	Mar 2022	$624M	Five of nine validator keys stolen; unnoticed for six days [25]	Delayed, capped, held
Poly Network	Aug 2021	$611M	A cross-chain message replaced the bridge's keeper key [26]	Capped, held
BNB Token Hub	Oct 2022	$586M	Forged proof minted 2M BNB; the whole network was halted [27]	Capped, held
Bitget	Sep 2026	$352M	Spoofed transaction data triggered hot-wallet approvals [28]	Delayed, capped, held
Wormhole	Feb 2022	$326M	Signature check bypassed; 120,000 unbacked wETH minted [29]	Capped, held
Liquid	Sep 2026	$320M	Validation bug minted unbacked L-BTC, withdrawn within 36 minutes [30]	Capped, held
KelpDAO	Apr 2026	$292M	A single bridge verifier accepted a forged message; released rsETH posted as lending collateral [31]	Capped, held
Drift	Apr 2026	$285M	Admin approvals pre-signed over a week earlier; fake collateral listed; drained in 12 minutes [32]	Blocked, capped
WazirX	Jul 2024	$235M	Signed data differed from the custody interface; wallet upgraded to the attacker's [33]	Blocked, delayed, held
Cetus	May 2025	$223M	Faulty overflow check; $60M bridged away before the pause [34]	Reverted, capped, held
Euler	Mar 2023	$197M	Donation function skipped the health check [35]	Reverted, capped, held
Nomad bridge	Aug 2022	$190M	A faulty upgrade trusted the zero root; hundreds of copycat withdrawals [36]	Capped
Beanstalk	Apr 2022	$181M	Flash-borrowed votes passed a malicious proposal in one transaction [37]	Capped
Balancer	Nov 2025	$128M	Rounding error present in the code since 2021 [38]	Reverted, capped, held
Multichain	Jul 2023	$126M	Every bridge key under one person's control [39]	Delayed, capped, held
BadgerDAO	Dec 2021	$120M	Injected website script requested unlimited approvals [40]	Blocked
Mango Markets	Oct 2022	$114M	Collateral price pumped on the venues its oracle used [41]	Capped
Harmony bridge	Jun 2022	$100M	Two of five multisig keys stolen [42]	Delayed, capped, held
Orbit bridge	Dec 2023	$82M	Seven of ten signers compromised; drained in 17 minutes [43]	Delayed, capped, held
Curve pools	Jul 2023	$62M	Compiler bug disabled reentrancy locks [44, 45]	Reverted, capped

Table 2: *The largest exploits and how Joltzu responds. Losses are in US dollars at the time.* Blocked*: the attack's mechanism is unavailable under Joltzu's rules or blocked by default.* Delayed*: with account safeguards, withdrawals beyond the daily allowance and changes to the signers wait for a delay in which a canceller can stop them.* Reverted*: an invariant the protocol registers on its own accounting would have reverted the exploit transaction.* Capped*: outflow budgets limit what could leave.* Held*: a bonded pause or a Council hold can freeze stolen proceeds while they are still in an account on Joltzu, though not those already swapped, bridged or dispersed.*

7.2. Stolen keys and deceived signers

By value, most losses came from compromised signing and not from broken code. Let's first explain what happened at Bybit. The signers approved what their interface showed as a routine transfer, while their hardware wallets blind-signed a call that replaced the multisig wallet's implementation with the attacker's [2, 47]. WazirX lost $235 million the same way, and in September 2026 spoofed transaction data deceived Bitget's approval process [33, 28]. At Drift, attackers persuaded the council signers to pre-sign hidden administrative transactions which stayed valid for more than a week, and there was no timelock to delay their execution [32]. Ronin, Harmony, Orbit and Multichain lost the keys securing their bridges outright, and Ronin's theft went unnoticed for six days [25, 42, 43, 39].

In Joltzu we remove the mechanisms these attacks relied on and limit what remains:

- A native multisig account has no implementation contract to replace, and changing its signers is an explicit administrative action. Approvals are tied to the complete decoded action, the policy version, a nonce and a short expiry, so a pre-signed approval cannot stay valid for a week (Section 3.3).
- With code eligibility enforced, an attacker cannot deploy

the malicious contract that Bybit's signers were tricked into calling (Section 5.1).

- With safeguards, even a change of signers that the signers really approve, which is what Bybit's and WazirX's swapped implementations amounted to, waits for the account's delay, and withdrawals beyond the daily allowance wait too, so a canceller has hours or days to stop them. Ronin's, Harmony's, Orbit's and Multichain's stolen keys would have been able to sign, but not to take more than the allowance before someone could cancel (Section 3.5).
- Exchange and bridge accounts move large sums under outflow budgets and pre-arranged allowances tied to destinations (Section 5.2). Ronin's 173,600 ETH and 25.5 million USDC left in two transactions, and a budget would have capped both and raised an immediate alarm.
- Stolen proceeds can be frozen wherever they sit on the chain (Section 5.7). Bybit's attackers split the funds within two hours and finished moving them in about ten days, while less than 4% was ever frozen [48, 49]. On Joltzu anyone tracing those funds could have paused each account as they found it by posting a bond, without waiting for an issuer or a committee to respond, and the Council could then have held what was caught.

7.3. Bridge verification failures

Bridges have lost more than any other kind of protocol, and in 2022 they accounted for 64% of DeFi hacking losses [50]. Wormhole, BNB Chain's Token Hub and Liquid each had a verification bug which let attackers mint unbacked assets [29, 27, 30]. A faulty upgrade made Nomad trust the zero root, which let hundreds of copycats withdraw [36], and a single message to Poly Network replaced its keeper key [26]. KelpDAO's bridge relied on one verifier, which accepted a forged message, and the attacker immediately posted the released rsETH as lending collateral and borrowed real ETH [31].

In each of these the outcome came down to speed. Liquid's unbacked bitcoin was withdrawn within 36 minutes, and while BNB Chain stranded most of the 2 million BNB the attacker minted, it could only do that by halting its entire network [51]. On Joltzu, each registered bridge has its own outflow budget, which means even a complete verification failure, including Nomad's copycat rush, is capped (Section 5.6). Anyone who notices unbacked funds can pause them by posting a bond, although an attacker who moves straight away, as KelpDAO's did with its collateral step, can get there first, and in that case it is the budget that limits the damage. Security agents compare bridged supply with locked backing, and an issuer committee can hold unbacked balances without stopping the network.

7.4. Application bugs, oracles and governance

Application exploits are the most frequent incidents. For instance, Euler's donation function skipped a health check [35]. A faulty overflow check let Cetus's attacker withdraw real reserves for a one-token deposit, and about $60 million was bridged away before the protocol paused [34]. Balancer's rounding error had been in its code since 2021 [38], and a Vyper compiler bug silently disabled the reentrancy locks of Curve pools built with the affected versions [44]. An attacker drained Mango Markets by pumping the price its oracle used [41], and Beanstalk fell to flash-borrowed voting power that passed a malicious proposal in a single transaction [37].

Joltzu screens code before deployment and re-screens deployed code when a new bug class appears, such as a flawed compiler version. A protocol can also register an invariant over its own accounting, which validators check at the end of every transaction that touches it (Section 5.3). Euler's attacker finished with debt its collateral did not cover, and Cetus, Balancer and the affected Curve pools ended up holding less than their shares were owed, so an invariant comparing those quantities would have reverted each exploit whatever the bug behind it, although it would not have helped Mango or Beanstalk, whose attacks followed the protocols' own rules. Funds can only leave a protected protocol as fast as its outflow budget allows, and bonded pauses and holds can stop stolen proceeds that have not yet moved on. Native lending accepts only approved collateral, priced by feeds with independent sources and liquidity requirements, and a manipulated or stale price has no way to raise withdrawal capacity (Section 12.7).

7.5. Malicious approvals

Some attacks do not need a stolen key or a code bug at all, only the victim's approval. BadgerDAO's website was altered to request unlimited token approvals, and more than 500 addresses granted them [40]. Wallet drainers took $494 million in 2024, with permit signatures (token approvals signed off-chain) behind most of the largest losses [3]. Wallet-side defences have not kept pace, because a warning only protects the users of the wallet that shows it, can be dismissed and does nothing to limit what a signed approval later takes.

Joltzu Wallet caps allowances and ties every grant to a verified website origin and its registered contracts. Before the user signs, it decodes permits and other signature requests and flags approvals to new spenders (Sections 8.3 and 8.4). Validators then enforce the grant's limits on every spending path, so a user who is deceived once still loses at most what the grant allows.

7.6. How networks respond today

When exploits happen, networks improvise. BNB Chain halted. Sui's validators blocked the Cetus attacker's addresses and later voted through an upgrade to move $162 million [52]. Arbitrum's Security Council upgraded a core contract to impersonate the KelpDAO exploiter and move 30,766 ETH [53], and in October 2026 it forced two transfers out of the Binance attacker's wallet [46]. After the Tectonic exploit, Cronos halted and rolled back 10,961 blocks, nearly

two hours of history [54]. Each of these interventions recovered funds, but through powers which users had not been told existed. We would rather users knew in advance, which is why Joltzu spells out up front how it contains an exploit, using holds with a recorded scope and review, bonded pauses and outflow budgets. It does not halt the network, roll back blocks or impersonate accounts.

7.7. AI-accelerated exploitation

AI models are making exploits cheap to find. On SCONE-bench, a set of 405 contracts exploited between 2020 and 2025, AI agents produced working exploits for 207 (just over half), worth $550 million in simulated funds. On incidents after the models' training cutoff they exploited 19 of 34, worth up to $4.6 million [55]. Scanning 2,849 recently deployed contracts at $1.22 per run, the agents found two new vulnerabilities, and a real attacker independently exploited one of them four days later. Simulated exploit revenue doubled every 1.3 months over that year, and every 0.7 months for the newest models in 2026 [56].

Independent work agrees with this. The A1 agent exploited 63% of the real vulnerable contracts in its benchmark, and its authors estimate that attackers break even on exploits worth $6,000 while defenders need $60,000 [57]. A GPT-4 agent exploited 87% of a set of disclosed but unpatched software vulnerabilities from their descriptions alone [58]. Leaving bytecode unverified doesn't protect a contract either, because decompilers and language models now scan it at scale [59]. The bottleneck has moved from finding vulnerabilities to fixing them. For instance, Anthropic's Project Glasswing found more than ten thousand high- or critical-severity vulnerabilities in a month, yet only 75 of the first 530 reported to open-source maintainers had been patched [60].

Old protocols. Contracts deployed years ago were written and audited with human attackers in mind, and each newly discovered bug class can now be checked against all of them at once. Balancer's flaw dated from 2021, and Truebit lost $26 million in 2026 through an overflow in a contract deployed in 2021 [38, 61]. Old contracts like these cannot be patched, which means protection for such code has to come from outside it. On Joltzu, outflow budgets, bonded pauses and holds apply to old code as much as to new, and the AI Agent Network re-screens deployed contracts as models improve.

New protocols. Attackers scan new code as soon as it is deployed. A1 estimates that an attacker succeeds 86–89% of the time when a vulnerability is found immediately, but only 6–21% of the time after a week's delay [57]. This means the race is decided in the first days. Joltzu moves that race to before deployment, where projects are generated with tests and invariants, screened with the same class of models that attackers use, and only deployed once admitted, and the invariants can then be registered so that validators enforce them on every transaction (Section 5.3). New protocols can also start with conservative outflow budgets that grow with their operating history.

Security procedures. We assume that attackers will find some vulnerabilities first, quickly and cheaply, as any network that holds real value has to, and our procedures follow from that:

1. Screen every contract with frontier AI models before deployment, alongside human review and formal checks.
2. Re-screen deployed code continuously as models improve and new bug classes appear.
3. Treat signing as the main attack surface, using decoded approvals, short expiries and separate keys.
4. Contain exploits by default with outflow budgets, safeguards, invariants, bonded pauses and bounded holds, so that a successful exploit takes less and its proceeds can be stopped before they are dispersed.
5. Specify emergency powers and their limits in advance, to keep anyone from improvising during an incident.

8. Joltzu Wallet

Joltzu Wallet is our own wallet, and we built it to close the gaps in existing wallets described in Section 7. It shows what a transaction will do before it is signed, blocks high-risk requests it cannot decode and binds every permission to a verified website and a budget. It comes as a browser extension and a mobile app which share one account model, AI agent, permission manager, messenger and identity controls, and onboarding explains signers and recovery. Applications request named actions with budgets and durations, so routine use within an approved grant needs no repeated popups. Mobile app links and QR requests also identify the requester before approval. Both apps show balances, bridge transfers, activity, active holds and permissions, and they can fund a pending action through the payment rails in Section 9.4. Adding a device needs the account's authorization, while encrypted synchronization and encryption-key recovery stay separate from device pairing and signing-key custody.

Joltzu Wallet also supports lattice-based cryptography, in the form of ML-DSA-44 signatures (Section 16.2). A user of the browser extension can opt in to a post-quantum account, which has no secp256k1 key at all, signs native transactions with its ML-DSA key and pays its own fees, and an existing address can move to that key and keep its balances, which retires its old key. Because this is a software key, the wallet stores its seed encrypted at rest, opens it for one reviewed transfer only after the user has verified themselves, overwrites it afterwards and doesn't let websites reach it. The trade-off is that there is no backup, which means losing either the browser profile or the credential that unlocks the seed loses the funds. These accounts send native JOLT and don't make contract calls or swaps, connect to applications or sign for services that take secp256k1 signatures, and this support covers signing, so the wallet's message encryption isn't post-quantum.

8.1. Built-in wallet agent

The wallet has an optional AI agent to help users understand their account and prepare actions. It explains onboarding, permissions, failures, balances and bridge status, and it can prepare payments, trades, lending actions, multisig proposals and messages. It starts read-only, and the wallet stays in charge by validating every call the agent proposes, keeping the signing keys itself and asking for approval before granting any new authority. The agent says what evidence its answers rest on and how uncertain they are, and it treats external content as untrusted. If it uses remote inference, private inputs need explicit consent and the wallet shows what it costs.

8.2. Permissions without repeated popups

To avoid a popup for every action, an application asks for permission on named contracts, operations, assets and recipients, within time and spending limits [62]. For instance a game might receive permission to spend 20 JOLT per day and 100 JOLT in total (so 5 days of spending at the full daily rate) until an expiry, and both caps apply across concurrent requests. New recipients, upgraded implementations or broader actions need fresh consent. Grants keep working while the wallet is closed, and the permission manager shows each grant's remaining budget, expiry and activity and lets the user reduce or revoke it. Device access, such as camera permission, comes with no spending rights.

For token approvals, the wallet checks the asset, spender, implementation, allowance and expected debit, then applies a bounded approval or an atomic approve-and-use flow, while direct JOLT payments need no allowance at all. Budgets count outstanding allowances as exposure and reconcile consumed allowances without double-counting. Before signing, the wallet checks the asset's Council approval, KYC requirements, upgrades, restrictions and the user's grants, then decodes and simulates the call. By default it blocks high-risk calls it cannot decode, and a screening override cannot widen a signed grant or get around protocol restrictions.

8.3. Domain registry and phishing protection

To protect users from lookalike websites, the Domain Registry binds each verified website origin to the accounts and contracts that control it, and the wallet checks for Unicode confusables, lookalike letters, misspellings and misleading subdomains [63]. For example, `joItzu.com` (with a capital I) normalizes to `joitzu.com`, not `joltzu.com`, because DNS ignores case [64].

Registering a domain takes a mailbox challenge sent to an email address at the domain, with authenticated alignment [65], and also a DNS TXT record or an approved HTTPS proof. The owner signs the domain, account, nonce and expiry. Services attest to these checks and validators verify the attestations, which is why the validators themselves do not contact email or DNS servers. Records bind contracts, implementations and dates, and ownership changes need revalidation. A registration only establishes an association and doesn't tell you that the website is safe.

8.4. Website-bound permissions

Each grant ties the exact website origin and registry version to the chain, implementation, spender, assets, operations, arguments, limits and expiry. This means a grant doesn't carry over to another site that uses the same contract, and native apps and agents need their own grants. The wallet captures the origin from the protected browser, app-link or QR context, while validators can only verify the signed association. NFT spending by a new site and collection-wide approvals get flagged, while AI review weighs contract age, domain history and simulated asset movements, and a proxy upgrade needs a fresh assessment instead of inheriting its old reputation.

Validators enforce the resulting permissions on direct sends, calls, delegated spending and token pulls [66]. Function selectors alone are not enough when calldata or proxies can reach other operations. Disconnecting a website does not revoke token allowances, while an on-chain revocation takes effect once it is confirmed and global ordering resolves any race (Figure 9).

8.5. Contacts and messaging

The wallet's authenticated, end-to-end encrypted messenger supports favourite contacts, presence, payment requests, blocking and spam controls, although routing metadata may remain visible. Messaging keys and relay retention are separate from signing keys, and reading a message grants no spending authority. The wallet resolves Joltzu aliases, ENS names and Unstoppable Domains names through supported records [67, 68], then shows the resolved destination and flags any change, although a record for another chain isn't automatically the correct Joltzu address.

8.6. Bridges and funds in transit

To show users where their funds are, the wallet tracks each bridge transfer through source-chain finality, bridge processing and destination settlement, and it shows the amounts, fees, transaction links and recovery status. Value in transit is counted once. Tracking a transfer cannot tell you whether the bridge is secure. Transfers between accounts assigned to different validator groups need no bridge, because all groups share one state.

8.7. Automation while the wallet is closed

Subscriptions and conditional callbacks can be processed while the wallet is closed, either by a JSN keeper (Section 13) or by any other authorized executor. On-chain execution still rechecks grants, conditions, gas and available funds. When the wallet reopens, it shows deadlines, pending actions and any failures.

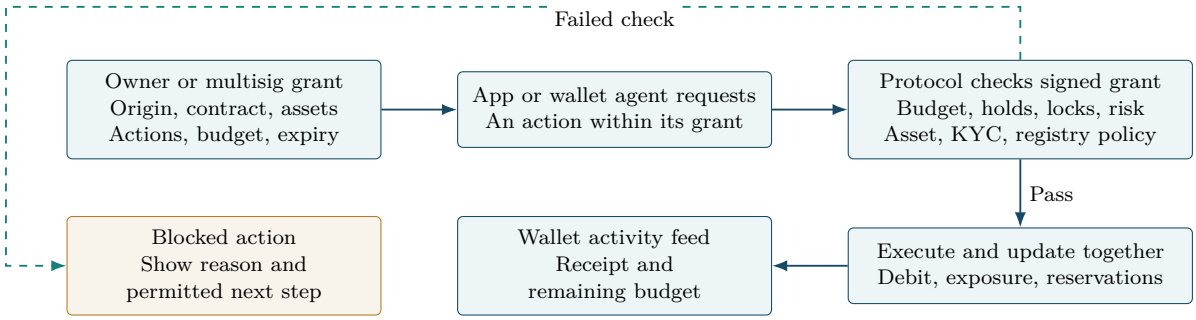


Figure 9: *Both wallet apps capture who is making the request and obtain a bounded grant. Website grants are bound to the origin and registered contract, and separate agent grants are bound to their executor. At execution the protocol checks the signed scope, spending limits, asset and KYC policy, registry status, locks and holds.*

8.8. Direct advertising and participation rewards

Joltzu Wallet has an opt-in inbox which lets approved advertisers pay users for defined participation, following precedents such as Brave Rewards [69]. No essential wallet function requires ads, and users control the categories, frequency and senders. Campaigns have separately funded budgets, eligibility rules and expiry, and matching favours on-device preferences, while ads are isolated from approvals and have no access to keys, credentials or spending authority. Joltzu limits abuse with unique receipts, rate limits and appeals, although a playback signature cannot prove that a human actually paid attention. Accepted rewards can be paid through funded vouchers (Section 9.5), and a user who opts out keeps the claims they have already earned.

9. Payments

Joltzu's payment features extend account authority to everyday commerce, which covers who pays the fee, how several steps combine into one action, how subscriptions run, how a pending action gets funded and how small or offline payments settle.

9.1. Fee sponsorship and payment in other assets

A sponsor can co-sign a user's exact transaction and pay its JOLT fee, and doing so gives the sponsor no authority over what the transaction does [66]. A sponsor can also quote the fee in another listed asset, such as USDC, in which case the user signs the asset, rate and maximum charge. That maximum is escrowed before execution and any unused amount is returned, while conversion and service charges are disclosed separately. A multisig account can also repay whoever relays its approval with a fixed relay fee in JOLT or USDC (Section 3.3), which suits co-signers and exchanges that submit on the owner's behalf. Unlike Arc [70], Joltzu doesn't smooth fees against swings in demand, because per-service budgets (Section 4.7) already keep one busy application from raising the price for everyone else, and a fee that lagged behind a surge would let spam through cheaply.

9.2. Atomic transaction plans

An atomic plan combines several calls into one all-or-nothing action, for instance redeeming a lending claim for cash, exchanging it and paying a merchant [71]. The account's approval binds the chain, account, policy version, nonce, a short expiry and the exact list of calls, each with its target, value, gas limit and calldata. The calls execute with the account as sender and are still subject to current permissions, holds, locks and outflow limits. If any call fails, all of the plan's effects roll back.

Fees are bound separately in the transaction envelope and are paid by the account or a sponsor. This means approving a plan doesn't by itself authorize a fee debit (Figure 10). Each call carries its own limits, such as a swap's minimum output. Oracle and cross-chain steps remain asynchronous.

9.3. Recurring payments

A recurring mandate fixes the merchant, asset, amount per period, interval, number of payments and end date. Mandates can pay in JOLT or any approved asset. Anyone can execute the payment currently due, and typically this will be a JSN keeper. Each period can only settle once, and a missed period gets skipped instead of building up, so an outage never produces a surprise catch-up debit. Changing a mandate needs the owner's consent. Cancelling it, or making any change to the account's policy, stops future collections without erasing history or open disputes.

9.4. Wallet-managed payment rails

When an account lacks the funds for an action, Joltzu Wallet offers integrated card, bank-transfer or cryptocurrency on-ramps and shows the asset, provider, quote, fees and expected availability. It then waits until the funds are actually spendable, which includes any bridge requirements, before it revalidates and simulates the original call. If the terms have materially changed the user has to consent again, and if the action gets cancelled the funds stay in the account.

Applications do not need a provider integration or any special contract interface. Funding and execution are separate stages. A card authorization is not on-chain settlement,

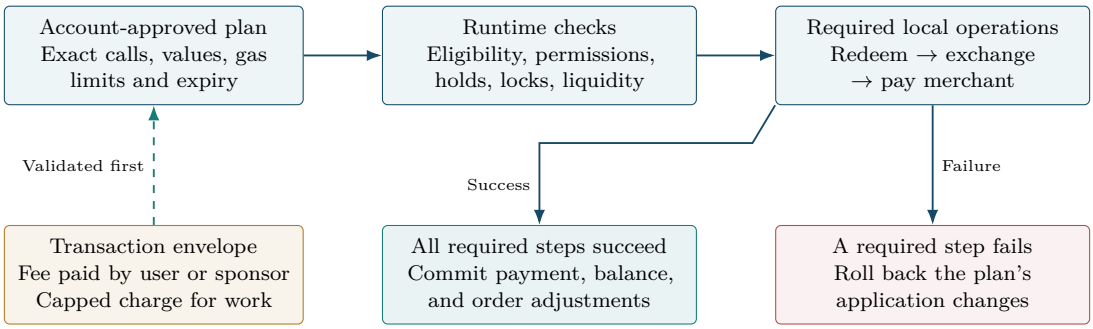


Figure 10: *An atomic plan combines admitted applications and native services. The wallet previews the sequence, and execution checks its limits against the current state.*

and a later provider dispute does not reverse finalized application state [72, 73]. Status tracking and reconciliation prevent duplicate submissions, and availability depends on the provider and region.

9.5. Funded vouchers and offline spending

Many payments are too small or too frequent to settle one at a time, such as game actions, API calls, articles, sensor readings or purchases at a festival with poor connectivity. Funded vouchers are how Joltzu handles them. The payer moves a fixed budget of JOLT into a vault for one recipient, naming a voucher-signing key and a claim deadline, and then signs vouchers for increasing cumulative amounts. The recipient collects these off-chain, over SigNet (Section 13.6), by QR code or on a local network, and can redeem them at any time before the deadline [74, 75].

For latest cumulative amount A, already settled amount S and budget B,

$$\Delta = \max(0, A - S), \qquad S \le A \le B. \tag{4}$$

Only the increase Δ is paid, which is why an older voucher pays nothing. For example, 100 actions at 0.01 JOLT settle as one 1-JOLT voucher. Anyone can submit a claim, but it only pays the fixed recipient. A batch checks each voucher separately, so an invalid item leaves its vault unchanged, while a failure of the outer transaction reverts the whole batch (Figure 11).

Because the budget is funded in advance, a voucher is as good as cash up to the vault's remaining capacity, and it stays valid if the payer later changes keys or recovers the account. To withdraw, the payer requests revocation. Claims can still be made until the earlier of the original deadline and one hour after the request, and after that the payer can reclaim the remainder. This means the recipient always has time to redeem vouchers already received. If a voucher key is compromised, all it can do is pay the fixed recipient, within the remaining budget. Settlement still passes locks and outflow limits.

The same mechanism works for offline merchants such as airlines or festivals, where a customer funds a vault for the merchant and signs vouchers at terminals without a connection. Terminals cannot tell on their own when a voucher has been copied to another terminal, so guaranteed acceptance needs a coordinator, secure hardware or a separate budget per terminal, although the ledger does reject any over-claim on reconnection. We plan to add vaults in approved assets. The wallet distinguishes funded, pending, settled and refundable amounts. Vouchers carry no identity records, but they do reveal amounts and counterparties.

10. Escrow, Conditions and Inheritance

Some payments should only happen when something else does, for instance when a delivery arrives, a deadline passes, a price is reached or an owner stops responding. Joltzu provides these as funded native services. This means the payer's funds move into the service when the agreement is made, and from there the service can only release them to the named recipients or back to the payer.

10.1. Escrow with delivery confirmation

Before payment, the buyer, merchant and adjudicator all accept the exact terms, including evidence, deadlines, fees and fallback. The buyer confirms delivery by scanning an item code and signing a receipt bound to the chain, purchase, item, recipient, escrow and nonce, and the code on its own authorizes nothing. For a receipt accepted at time t_a,

$$\begin{aligned} \text{Release} = \text{ValidReceipt} \\ \wedge\, (t \ge t_a + 7\,\text{days}) \\ \wedge\, \text{ClearDisputeStatus}. \end{aligned} \tag{5}$$

An open dispute blocks release, and the adjudicator can award any split before its deadline, although if it does not act the buyer is refunded. Without a receipt, the agreed delivery deadline and fallback apply. Disputed funds are not available for lending or orders, and a refunded amount cannot also go to the merchant (Figure 12). Ordinary finalized

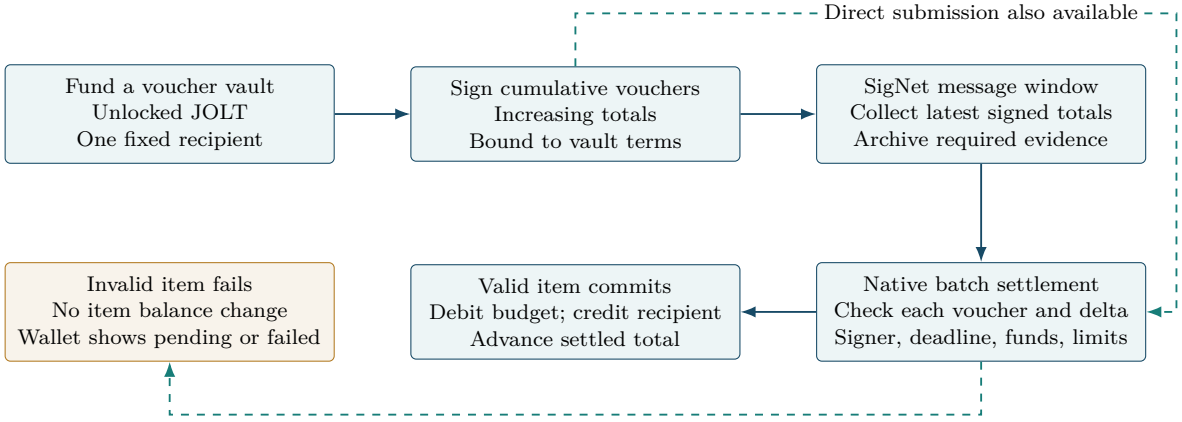


Figure 11: *Funded vouchers settle many small payments at once. Vouchers can travel over SigNet or go directly to the recipient, and a claim only pays the increase over the total already settled, so a repeated voucher can't pay twice.*

transfers work differently, and they are never reversed on request.

10.2. Composed conditions

More general conditions are trees of threshold gates, where each gate means "at least k of its n inputs", built over four kinds of evidence: a time being reached, a hash preimage being revealed, an oracle observation crossing a bound and a threshold of named signers attesting to an event. `AND` and `OR` are the special cases $k = n$ and $k = 1$. Anyone can submit evidence, and because valid evidence is latched permanently, a condition which has been satisfied stays satisfied. An oracle leaf only sees the feed's value at the moment evidence is submitted, with no continuous history behind it. A conditional payment has a deadline, after which the payer can reclaim it. Condition trees have no negation or veto, so protection against a bad outcome comes from choosing the evidence and signers carefully. AI can help express a rule, but execution only evaluates the resulting typed condition.

10.3. Inheritance

An owner can fund a bequest, which is an amount reserved for up to sixteen beneficiaries with fixed shares totalling 100%. If the owner does not check in for the inactivity period, for example two years, anyone can start a notice period, such as 30 days, during which the owner can still check in or cancel. When the notice ends, anyone can settle the bequest to its beneficiaries. Only an explicit check-in by the owner resets the inactivity timer, so incoming transfers, relayer activity and delegated payments have no effect on it.

The owner can also name verifiers, such as family members or a solicitor, and a threshold of them can report the owner's death with a commitment to their evidence. This starts the notice period at once, without waiting for the inactivity period to run out. Starting the notice is all verifiers can do. They cannot redirect payments, change shares or settle early, and an owner check-in voids their report. Their signatures establish who reported a death but not the death itself, and an AI classification or a document hash on its own will not establish it either. The Council has no authority over estates.

A bequest only covers its funded amount and not every asset the owner holds. Settlement respects locks and holds, and once a bequest has settled it cannot be redirected. Legal estate requirements still apply off-chain.

10.4. Vesting grants

A vesting grant is a funded promise, like escrow and a bequest. The grantor locks an exact amount of JOLT for one beneficiary and can mark the grant revocable when creating it. Nothing is released before the grant's cliff, at which point the share accrued so far becomes available and release then continues linearly to the end of the term. Anyone can trigger a claim, but it only pays the beneficiary. The grantor can stop a revocable grant once, and when that happens the unvested remainder returns to the grantor while amounts already vested stay claimable. A grant only releases what its grantor funded, and it has no way to mint JOLT.

11. Identity and Compliance

Some assets and services need to know who is allowed to use them, for instance tokenized stocks, regulated payment assets and fiat on-ramps. Joltzu supports this with verified credentials which are attached to accounts and checked at execution, and none of this puts identity documents on-chain.

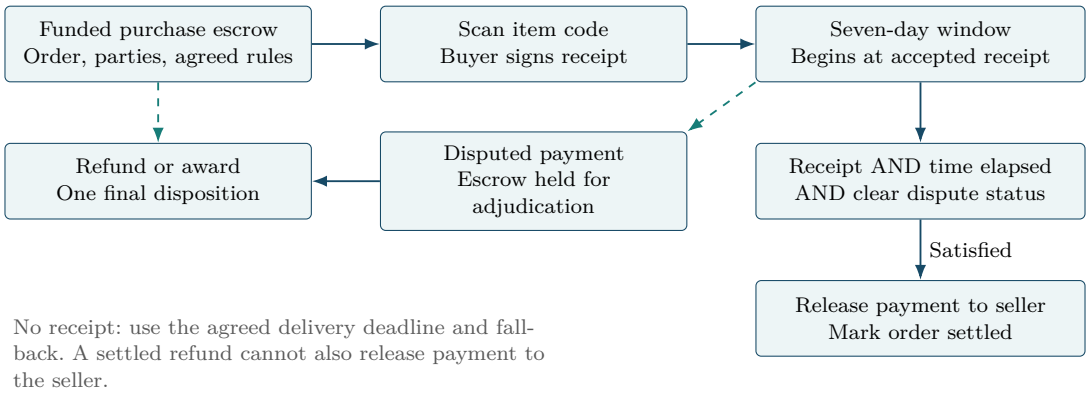


Figure 12: *Delivery settlement combines a receipt signed by the buyer with a dispute window. The barcode identifies the order, and a contested payment follows the agreed adjudication path.*

11.1. Verified credentials

A native `AddKYC` operation binds a credential from an accepted issuer to the owner's authorized account [76]. It records the credential's schema, version, validity, status and a commitment to its contents, but not the underlying documents. Validators verify the issuer's signature and the credential's eligibility, which is why a user declaring themselves eligible isn't enough. An attestation of age or residency can stand in for the details behind it. Issuers have to get the user's consent before issuing, and if an issuer later revokes a credential it becomes invalid without any history being rewritten.

11.2. KYC-gated operations and regulated applications

Restricted services check current credentials, holder class and jurisdiction at execution, and for a trade this includes both counterparties. Attestations bind the account, issuer, scope, version, validity and committed status, so validators have no need for decrypted documents or live queries to providers. Transfers, delegated pulls, lending and intermediary routes enforce the relevant beneficiary policy. If a credential expires or is revoked, any ineligible orders get suspended, and refunds and redemption follow the asset's recovery rules.

11.3. KYC banking on- and off-ramps

Supported providers convert fiat currency to JOLT or approved assets and back, while the wallet discloses identity requirements, fees, limits and settlement status, and each provider decides which credentials it accepts. Identity and payment are separate permissions, and providers handle reversals and refunds without rewriting finalized history. We are not implying any provider partnership here.

12. Markets and Lending

Joltzu's markets are native services of the protocol and are not independent contracts sitting on top of it. The orderbook, automated market maker (AMM) pools, firm quotes and lending all share the account ledger, which means a trade, loan or payment settles atomically against the same balances, permissions and safety checks, and any conflicting commitments get resolved in the same commit.

12.1. Asset approval

Being ERC-20 compliant is not enough on its own to make an asset eligible [77]. The Security Council reviews its code, issuer, backing, custody, redemption, privileges and holder restrictions, focusing on tokenized stocks, verifiable claims and payment assets, and a stock token has to identify real contractual or ownership rights. Approval binds the asset to its implementation, issuer and policy, and registers its markets with their precision, quote assets, tick and lot sizes and any KYC policy. Factories, wrappers and upgrades have no way around the review. Once an asset is listed, its issuer committee manages holder permissions, holds and delisting (Section 5.5). Approval doesn't guarantee price, liquidity or issuer solvency.

12.2. Orderbook and perpetual futures

The native orderbook matches orders by price-time priority, and each fill executes at the maker's price and may be partial. A resting order holds a soft reservation on the maker's cash. If that cash moves elsewhere the same commit resizes or cancels the conflicting order, which is how Joltzu makes sure an order never promises funds the account no longer has. Matching and settlement recheck eligibility, KYC and restrictions, and a restriction can block fills while still allowing cancellations. Listing on its own does not create liquidity.

An asset approved for spot trading is not automatically eligible as collateral or for derivatives.

12.3. AMM pools

Constant-product pools complement the orderbook, and Joltzu runs Uniswap V2-compatible pairs natively [78]. They keep V2's pricing, fees, liquidity provider (LP) tokens, flash swaps and deterministic CREATE2 addresses, which means

existing V2 routers and integrations work unchanged. Pools accept only listed assets. We chose not to make pool creation permissionless, which is why pools are created through an administrative factory, beginning with JOLT/USDC. Pool reserves are native service accounts, which pauses and holds cannot name, so pools keep settling during an incident. Pools can be paired with a native farm that pays JOLT to staked LP tokens, and the JOLT/USDC pool has one from the chain's first day. Its rewards come from two sources, newly minted JOLT at a rate that only ever decreases and a share of protocol revenue (Section 3.9), so the farm's yield leans more on real usage as the minting falls away. Its owner can add pools and set allocations but has no way to raise the minting rate or touch anyone's stake, and stakers can always withdraw.

12.4. Firm quotes and currency settlement

A market maker can sign a firm quote that binds the assets, per-lot price, amounts, permitted taker, identifier, expiry and fees [79]. Because settlement is atomic, a merchant can receive one approved currency while the customer pays with another. Quotes allow partial fills. Unique identifiers prevent replay and remaining-amount records prevent overfilling, while global ordering resolves any races between fills and cancellations. A maker can back a quote with hard-reserved cash, which is released when the quote expires. Otherwise the quote depends on the maker's liquidity at fill time.

12.5. Lending

Native lending lets users supply JOLT for variable interest and borrow against approved collateral. Supplying creates an interest-bearing claim on the pool, recorded as non-transferable shares. This claim isn't cash, because it earns interest even while the JOLT behind it is on loan and it can be redeemed only against the cash the pool has available. Interest follows pool utilization, which is the fraction of pool funds on loan.

Borrowers post oracle-priced collateral within loan-to-value limits. Each loan has a fixed term, and a position becomes liquidatable once it breaches its liquidation threshold or reaches maturity. Whatever losses liquidation fails to recover fall on suppliers in proportion to their shares. Cash buffers support withdrawals, although at full utilization they cannot guarantee redemption. The rules for withdrawal priority and loss allocation also have to handle stale prices and restricted liquidations.

12.6. Funding orders from lending positions

A supply position can fund market activity directly. When a user places a bid or fills a quote from supply, the same commit redeems exactly the shortfall, up to a limit the user sets. This means the order is backed by real cash instead of a claim on money that has been lent out, while any supply that is not needed keeps earning interest. Cancelling an order returns its cash to the account, and the pool is not automatically resupplied.

A user can go further and let an order hold a cancellable reservation on the supply position instead, which redeems supply only when the order fills, so the whole position keeps earning interest until then. This isn't without cost, as settlement then depends on pool cash at fill time. If V_u is the user's supply value, H_u its hard encumbrances and C_a the pool's available cash, a debit x must satisfy

$$x \le \min\big(\max(0, V_u - H_u), C_a\big), \tag{6}$$

and locks, holds, fees, haircuts and account health can reduce this bound further (Figure 13). For example, a user supplying 1,000 JOLT might bid 700 JOLT for another asset. If another authorized action redeems and spends 400, the remaining position is 600. Because that no longer covers the bid, the same commit resizes or cancels the conflicting bid. Borrowing can also reduce pool cash without reducing anyone's claim, so every fill rechecks both position health and settlement liquidity. Reusing collateral needs a shared risk calculation so that nobody can pledge the same collateral beyond its permitted capacity.

The wallet shows supply value, accrued interest, available cash, collateral, debt and open orders separately, and previews which orders an action would resize or cancel.

12.7. Oracles and price feeds

The oracle network brings external observations into deterministic execution. It provides a JOLT price feed and a framework for other approved prices and events. A signature tells you who reported an observation, but how reliable that observation is depends on the quality, independence and manipulation resistance of its sources [80].

Since a signature doesn't tell you how good an observation is, each feed sets out what it reports and how, with its quote units, decimals, independent sources, aggregation, update frequency and liquidity requirements. For instance, several reporters repeating one venue do not count as independent sources. Each signed round then carries its observation and publication times, version, validity and quality. A report is tied to its exact context by binding the chain, service, feed and version, round or event, value, times and validity, and each publisher's sequence numbers have to strictly increase. Consumers need to know what state the data is in, which is why they distinguish proposed, disputed, finalized, stale and unavailable data and check freshness before trading, opening new risk or liquidating. A price and a one-off outcome are different kinds of question, and they have different bonds, evidence, adjudication and cutoffs. Corrections cannot silently rewrite settled state, and while AI may assess evidence it doesn't get to redefine the question.

Contracts read accepted reports through a native interface and execution consumes them deterministically, while collecting observations and running any supporting inference happen outside the transaction. Missing or manipulated data can never increase withdrawal capacity, and a

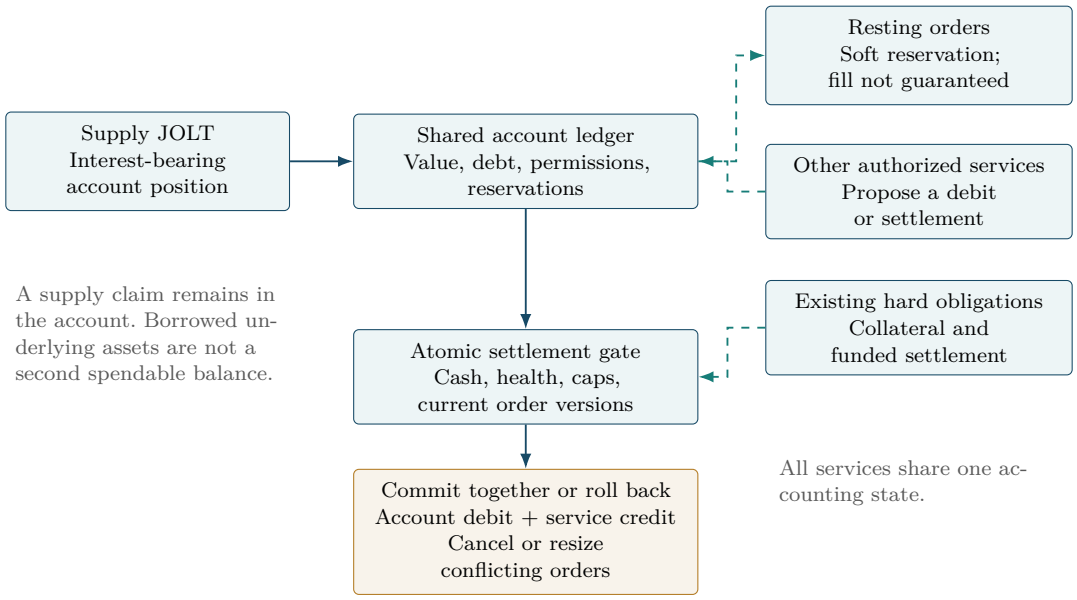


Figure 13: *Authorized services draw on a common account position. Settlement verifies available cash and obligations, then updates balances and conflicting orders atomically. Orders funded from supply redeem their cash when they are placed.*

feed reporting a price doesn't mean there is executable liquidity at that price.

13. Joltzu Service Network

The Joltzu Service Network (JSN) is the second part of Joltzu, alongside the protocol. It handles the work that cannot happen inside a transaction: hosting application frontends, serving chain data through public RPC endpoints, producing verifiable randomness (VRF), calling contracts through keepers when a condition becomes true or a payment falls due, co-signing transactions that meet an owner's rules, relaying signed messages over SigNet and running the AI agent network's workers (Section 6). JSN is distributed, not decentralized. Many machines provide its availability and capacity, with redundant gateways, competing keeper workers and replicated SigNet peers, but one operator runs them all.

We made JSN a separate project from the reference node and it runs apart from the validators. This means consensus doesn't depend on it, and if JSN has an outage its services can be delayed but block production and finality carry on. JSN publishes its latency measurements, and the protocol rechecks everything JSN submits, whether that is an ordinary transaction or a signed or proved result. Anyone who doesn't want to trust the operator can verify state and receipt proofs, run their own node or host an application elsewhere, and where the protocol allows it anyone can also execute a due payment instead of waiting for a JSN keeper.

13.1. Application hosting

JSN serves Studio releases (Section 14.3) from versioned, approved artifacts under the developer's deployment authorization, and developers can export a release to another host at any time.

13.2. RPC access

JSN operates the public node endpoints that wallets, applications and indexers use to read the chain and submit transactions. Developers create API keys in a JSN dashboard and connect over HTTPS or WebSocket, passing the key in the endpoint path or an authorization header. JSN only stores digests of these keys, and it treats a key in a URL as a secret which has to be kept out of every log.

The endpoints serve the reference node's Ethereum-compatible and native methods, including subscriptions to new blocks, logs and pending transactions. Because committed blocks are final, a dropped subscription resumes from the last delivered height with no reorganizations to handle, and answers about committed history can be cached. JSN's answers come from its own nodes and they are not proofs. An application that cannot trust the operator can request state and receipt proofs (Section 4.1) or run its own node.

Each request costs compute units weighted by its method. Every key has a rate limit and an optional monthly cap within its account's limits, and owners can tighten a key's limits or restrict it to chosen origins, source addresses, methods and contracts. An origin restriction stops other websites from using a key, although it doesn't stop other clients. Indexers, including those Studio generates, read the chain through the same endpoints and are metered like any other client, so JSN charges for the data an indexer reads and doesn't host the indexer itself.

Accounts pay for usage in bridged USDC (USDC.e) from a prepaid balance, and a deposit to the billing contract is

credited once its block is committed. Pricing is graduated. The rate per compute unit falls as an account's monthly usage passes published thresholds and each rate applies only to the units within its band. If a balance runs out the account drops to free-tier limits and doesn't lose access altogether. Rates, thresholds and limits will be published with the service.

13.3. Verifiable randomness

JSN supplies applications with verifiable randomness, which is separate from the consensus randomness in Section 2.2. A VRF job commits its request and gets back an output and a proof, the proof is verified on-chain before any contract consumes the output, and single-use identifiers prevent duplicate fulfilment. The VRF will be quantum safe, which rules out the elliptic-curve construction in RFC 9381 [81] (Section 16.2). The service is paid for in JOLT. Requests need timeouts and recovery, because a valid proof doesn't force anyone to deliver the output.

13.4. Keepers and callbacks

Anyone can register a keeper job for an eligible contract, triggered by a schedule, a state predicate, an event or a finalized oracle report. Registration fixes the target, calldata, expiry, gas budget and spending limits, an event job also binds the source chain, contract and required finality, and the owner can update or cancel the job later. When the job executes, the target rechecks conditions and authority, so observing an event gives nobody control over another account. Occurrence identifiers prevent duplicate completion, and the retries and fees for failed calls are bounded. Callbacks are ordinary transactions, which means congestion, restrictions or an exhausted budget can delay them.

13.5. Co-signing

On most chains a stolen key is all an attacker needs, because whoever holds it can sign from anywhere. JSN's co-signer gives an account a second signer which only signs when a request meets rules the owner has chosen, so a key on its own is no longer enough to move funds. Co-signing is optional and an account starts without it. The owner adds the co-signer's key to a multisig policy (Section 3.3), and a typical setup is two of three signers, made up of the owner's everyday key, the co-signer and a master key kept offline, such as a hardware wallet or a key held by a separate multisig. Everyday transactions are signed by the everyday key and the co-signer, and the master key is only needed when something goes wrong. Bots and payment services get the same protection when they run from their own accounts, which means a trading bot's key leaking from a server is not enough to empty that account either.

The owner can combine any of these rules. An IP allowlist limits which addresses or network ranges a request can come from, and an authenticator code requires the six-digit time-based code from an app such as Google Authenticator [82], whose secret is shared with the co-signer when the owner enrols it. The co-signer can also require a registered device, restrict the hours of the day, cap how much leaves each day and allow payments only to chosen recipients. Before signing, it decodes and simulates the request in the same way the wallet does and then signs the exact approval digest, which binds the chain, account, policy version, nonce, action, relay fee and expiry, so whoever relays the approval can neither change nor reuse it. IP rules work best for servers with fixed addresses, such as payment services, trading bots and exchange hot wallets, because home and mobile addresses change often and many customers of one provider can share an address. For people we would generally recommend an authenticator code or a registered device, ideally together with an IP rule, since a lookalike website can ask for a code and relay it to an attacker within the thirty seconds it is valid for.

Tightening a rule takes effect straight away, while loosening one, such as adding an IP address, removing the authenticator or raising a limit, waits for a delay of 48 hours in our illustrative parameters and notifies the owner, unless the master key signs the change as well. The co-signer treats an account policy change that removes a signer or lowers a threshold in the same way. This means an attacker who has the everyday key can't simply ask the co-signer to allow their own address. JSN keeps the rules off-chain, because publishing them would reveal where the owner signs from.

When the owner's ISP assigns a new IP address or their phone with the authenticator is lost, the master key approves the new address or authenticator immediately. If the co-signer is unreachable, the everyday key and the master key together satisfy the same two-of-three policy without JSN at all, and if the everyday key is lost the master key and the co-signer can sign while the owner sets up a new one. An account with no master key would have no way past a co-signer that stops answering, which is why the wallet asks for a master key or guardian recovery (Section 3.4) before it enables co-signing.

Once both signatures are collected, the co-signer can submit the transaction and pay its gas, and the approval repays that cost from the account through a relay fee in JOLT or USDC (Section 3.3), so the owner pays from their own balance as usual. An owner who would rather pay directly can submit from the everyday key's own address instead.

The protocol treats the co-signer as an ordinary signer, so validators check its signature like any other and co-signing adds no consensus rule. This means JSN's co-signer is only one choice, and users and exchanges can run the same co-signer themselves, with their own rules and their own servers, and list its key in place of JSN's. An exchange can, for instance, keep its hot wallets behind a co-signer inside its own network, which only signs withdrawals that its risk systems have approved, without trusting JSN or anyone else. The co-signer can't move funds by itself, since it is only one of the signatures the threshold needs, and the owner can remove it with an ordinary policy change signed by the everyday and

master keys. An owner can also name the co-signer as one of the account's cancellers (Section 3.5), so that it cancels a scheduled plan or change that breaks the owner's rules even when the other signers approved it.

13.6. SigNet

SigNet is JSN's temporary store for signed messages and events. Applications can, for instance, publish maker orders, votes or payment vouchers there without writing every message to contract storage, and indexing services such as The Graph can then serve these messages to dApps, which leaves them less dependent on each project's private database. Publishing costs no gas, and an account's message capacity is tied to its JOLT holdings. Messages are kept for up to two weeks, so applications that need records for longer have to archive them or keep commitments on-chain. SigNet detects duplicates with its own message sequence, which is separate from transaction nonces. A receipt from a SigNet peer only acknowledges the message and doesn't mean it has settled.

JOLT holders can delegate message capacity to other users without granting them any spending authority, and the capacity they delegate is bounded and expires. Imagine an election, where the votes do not need to be stored indefinitely so temporary storage suits them. With delegated capacity a newly created wallet can register and sign a vote without gas and receive evidence of inclusion. SigNet peers validate signatures, sequence messages, enforce storage quotas and may relay authorized payments, but they have no way to alter recipients or amounts. Vouchers carried over SigNet settle on-chain through funded voucher vaults (Section 9.5), and there is always a path to redemption other than SigNet.

14. Developer Platform

We want Joltzu's developer tools to take an application from an idea to a reviewed release without every team needing security expertise. AI generation, visual scripting, reusable plugins and engine integrations all produce ordinary, inspectable Solidity which goes through the same screening and eligibility checks as handwritten code. The tools build on both parts of Joltzu, because contracts run on the protocol while JSN provides hosting, RPC access and verifiable randomness.

14.1. From an idea to a specification

From an application brief, Joltzu's AI tools generate an inspectable Solidity project, including contracts, interfaces, wallet calls, indexing, tests and deployment files, and developers keep editable source and control of each release. The brief has to define the actors, permitted fund movements, deadlines, refunds and dispute rules before generation starts.

Example: build an escrow dApp

> *Build a milestone-escrow dApp using native JOLT. Buyers fund and approve each milestone; unapproved milestones can be refunded after agreed deadlines. Include no new token or administrator withdrawal. Generate the interface, tests and testnet deployment plan. Ask me to resolve missing payment or dispute rules first.*

The developer then approves testable invariants, which here are that deposits are conserved, releases are authorized and no milestone is both paid and refunded.

14.2. Generation, testing and release

Each generated project pins its templates, dependencies and compilers in a reproducible build manifest. Generated code and its generated tests can share the same mistake, so static analysis, fuzzing and bounded formal checks complement independent review. Reference tools include Slither, Echidna and SMTChecker [83, 84, 85]. Release review covers bytecode, privileges, initialization, dependencies and target network, and production deployment needs developer authorization and network eligibility (Figure 14) while developers keep their deployment credentials.

Review also covers frontends, backend access, data handling and wallet calls, and each interface has to match the reviewed ABI and addresses. A patch needs a reviewable diff, tests and fresh analysis, an upgrade needs authorization and immutable code needs a migration path [86].

14.3. Joltzu Studio

Studio saves developers from jumping between tools by bringing Solidity and interface generation, visual scripting, local execution, testnet deployment and release management into one workspace, and projects stay editable and exportable. Its components expose payments, vaults, markets, lending, identity, oracles and VRF, which saves wiring up each one, and wallet-managed funding (Section 9.4) needs no provider-specific integration. Before release, browser and mobile previews show permissions, KYC requirements, funding and simulated balance changes. Each release binds its build, dependencies, addresses, domains, flow profiles and reviews, and JSN hosts approved release artifacts (Section 13). Private input sent to a model provider requires consent.

14.4. Joltzu Visual Scripting

Smart contract development requires a specific skill and knowledge of blockchain, which makes custom contracts quite difficult to create. Joltzu Visual Scripting (JVS) reduces the impact of this by turning connected visual nodes into readable Solidity and EVM bytecode. Developers aren't stuck with the graph, because the source stays editable. Visual graphs and written briefs produce the same inspectable source and validation records. To keep costs visible, red storage nodes and gas estimates highlight expensive operations. Types include integers of different widths, booleans,

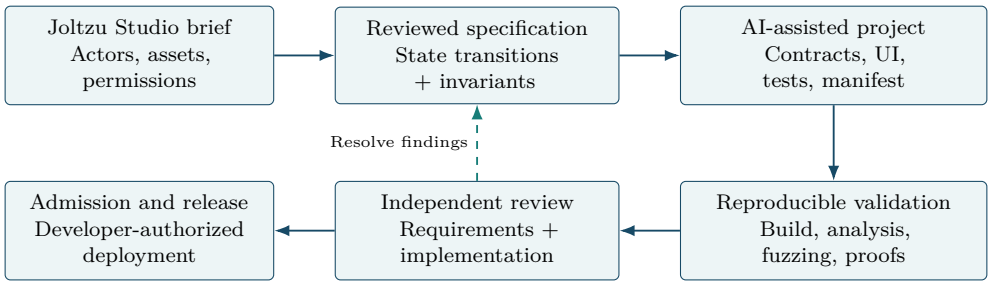


Figure 14: *Joltzu Studio takes an application brief through specification, generation, reproducible validation, independent review and admission. The developer authorizes the release, and the workspace has no way around network verification.*

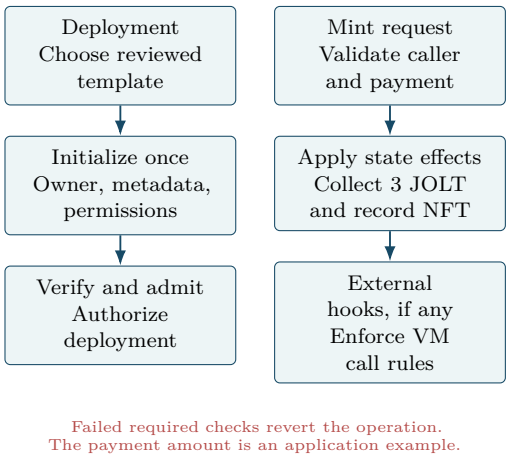


Figure 15: *Visual scripting for deployment and minting, including initialization, payment validation and failure handling. Generated contracts go through the ordinary admission process.*

addresses and structs, and automatic packing preserves their semantics. Generated flows use application guards, explicit failure paths and the checks-effects-interactions pattern. Nodes also expose plugins, precompiles and native trading, and Figure 15 shows deployment and an NFT mint costing three JOLT.

14.5. Reusable contract plugins

Each plugin instance is an ERC-1167 minimal proxy that points to a reviewed implementation, which means deploying one costs a fraction of a full deployment [87]. Plugins cover ERC-721, ERC-1155 and approved ERC-20 assets, and explorers show the shared source, although each instance still needs its own initialization and privilege checks. Plugin authors may receive disclosed usage royalties.

14.6. Aliases and pre-call hooks

Joltzu aliases are readable names held as ERC-721 tokens, similar to ENS, and the current owner sets the destination an alias resolves to. Transferring an alias points it at the new owner, so a name can't keep resolving to a previous owner. Wallets resolve aliases when preparing a transaction, and registration fees are protocol revenue (Section 3.9).

A contract can name its own pre-call hook, which is a bounded, read-only check that has to approve every call into the contract. This lets applications enforce allowlists without changing contract logic. A hook only implements the rules it was given and doesn't by itself establish legal compliance. The same service lets a contract name an invariant checker, which validators ask once at the end of each transaction that touched the contract instead of before each call (Section 5.3).

14.7. Game engine plugins

Games running in a browser can hook into some wallets, but desktop games don't have such luxuries. Our Unity and Unreal Engine plugins give browser, desktop and mobile games access to wallet connections, signing, contract calls, transaction status, randomness, payments, markets and JVS contracts. Keys stay with the player's signer, approved grants cover routine actions and broader spending needs consent.

14.8. Foundation grants

Foundation grants reward useful dApps, security work, accessible onboarding and sustained development, including fixes, integrations and documentation. Programs publish their budgets and eligibility, and rewards for long-term work can be paid through native vesting grants (Section 10.4). A developer doesn't need to issue a token to build a business.

15. Joltzu World

Joltzu World is a 3D multiplayer world played in the browser, in which Joltzu's features become places to explore. Instead of reading a dashboard, a player walks a robot avatar over to a reactor console to buy JOLT, to a trading floor to use the

orderbook or to an arcade table to roll dice with verifiable randomness, and every on-chain action goes through Joltzu Wallet. Like the developer platform, the world is an application built on the protocol and JSN. It runs on desktop and mobile with keyboard, mouse or touch controls, and there is nothing to download.

15.1. Areas

- **Power hall.** The reactor console opens JOLT purchases, and a vault displays the genesis NFTs a wallet has earned.
- **Trading floor.** This is the native orderbook and markets (Section 12), and the room shows live depth, fills and the player's own orders.
- **Arcade.** The dice table requests randomness from JSN. The player pays the fee and a keeper delivers the result with its proof, which the contract verifies before the dice land (Section 13). Arcade cabinets host partner games, starting with Estfor Kingdom.
- **Quarters.** Every player gets a room of their own on the mezzanine, with a desk showing their balances.
- **Atrium.** The central hall that connects the areas, with a large screen for announcements.

New players get objectives, collectables and warps to guide them through each area, which is how exploring the world doubles as an introduction to Joltzu.

15.2. Multiplayer and safety

Players join halls of up to ten through a lobby server, which keeps every avatar in sync about fifteen times a second, and they can use emotes, hall chat and global chat. The world follows the same safety principles as the rest of Joltzu:

- Players only connect to the lobby and do not connect to each other, which means nobody learns another player's IP address, and no wallet details are shared.
- The server assigns names, so nobody can pose as staff.
- Chat removes links, addresses and hidden characters before anyone sees a message, and each sender is rate-limited. A pinned notice reminds players that Joltzu never asks for funds in chat, and any player can be muted.
- The world never holds keys. Every purchase, roll or order is a transaction the player approves in Joltzu Wallet, which decodes and simulates it like any other.
- Shared events, such as another player's purchase, cannot be faked because they are read from the chain instead of from other players.

15.3. Wallet integration

Joltzu World connects directly to Joltzu Wallet. Because a popup for every small action would quickly get in the way of playing, a player can approve a limited grant for the world, for instance a daily JOLT budget for arcade fees, and routine actions then go through without repeated popups. Larger or new actions still ask for consent (Section 8.4). The grant is bound to the world's verified origin and contracts, and validators enforce its limits.

16. Governance and Post-Quantum Migration

16.1. Joltzu Autonomous Organization

The Joltzu Autonomous Organization (JAO) is Joltzu's counterpart to a decentralized autonomous organization (DAO). Through it the JOLT community proposes and votes on routine protocol upgrades and parameter changes on-chain, and approved changes have to pass policy checks before they execute. Joltzu deliberately keeps the JAO apart from the single-purpose committees of Section 5.6, so routine upgrades, Council incident actions and code admission each have their own authorization path. Committee membership, terms, appeals and key rotation are set through the JAO, which also decides how the treasury is spent (Section 3.9). Submitting a proposal or a bonded report doesn't give anyone the authority to edit code or seize assets.

16.2. Post-quantum cryptography and key migration

No machine can break today's signatures, but the estimates keep falling. In March 2026 Google put a break of secp256k1 at no more than 1,450 logical qubits and under 500,000 physical qubits, taking about 9 minutes from a primed state, and that assumes superconducting hardware with physical error rates of 10^{-3} [88]. Joltzu's 500 ms slots make it hard to intercept a transaction before it lands, as long as it's included promptly. The larger risk is recovering a key at rest, because an account's public key can be recovered from any signature it has made and validator keys are public by design. Our state, receipt and block commitments use Keccak-256 and don't need replacing.

So Joltzu uses post-quantum cryptography for everything it controls, and ordinary user accounts stay on secp256k1 because exchanges and standard wallets need it. Every consensus signature uses ML-DSA-44 [5], which is standardized, stateless and available in the OpenSSL version the node already pins. That covers votes, proposals, new-views, group voting, double-sign evidence and proofs of possession. The Security Council, the code-admission path and oracle publishers need ML-DSA signatures too, because their keys are set at genesis and no outside wallet is involved. Peer links use a hybrid key exchange (Section 17.11).

Joltzu uses neither BLS nor a hybrid of the two, which means a practical break of ML-DSA would allow forged finality with no second signature to stop it. We accept that risk, because ML-DSA is a NIST standard and keeping BLS would mean maintaining two key types for protection that ends once quantum machines arrive. Consensus names its signature types in one module and derives its size limits from them, so changing scheme stays contained. We also looked at leanXMSS, the hash-based signatures Ethereum is developing, and decided against them. The implementation is still an unaudited prototype [89], and reusing a one-time position breaks the scheme, while our signer may be asked for the same digest repeatedly. Proof-based aggregation doesn't pay at our committee size either. leanVM reports 318.4 KiB and

0.947 s to prove 1,290 signatures [90], which is longer than a slot, while a plain list from 22 signers is about 52 KiB and needs no proving. A signature list only reaches that proof size at about 130 signers.

Large signatures have a real cost, and we pay it in certificates. An ML-DSA-44 public key is 1,312 bytes and a signature is 2,420 bytes. These signatures don't aggregate, so a certificate lists one signature per signer and active committees are capped at 32. A commit certificate from 22 of 32 signers is about 54 KB, where the same BLS certificate would be about 536 bytes. With a block in every slot that comes to about 9.3 GB of certificates a day, and since finality archives are never pruned our storage budgets have to allow for it. With three phases and all-to-all delivery, each validator also handles about 465 KB/s of vote traffic, and verifying a round's votes takes about 7.4 ms on one core. These are calculations and not measurements, and running 32 validators inside a 500 ms slot across real networks is the thing most likely to change the committee limit. User transaction capacity doesn't change while accounts use secp256k1.

Ordinary accounts upgrade through the native policies in Section 3.2. This is the same idea as Ethereum's account abstraction, where an account says how it is validated so that the scheme can change while the account stays. An ML-DSA signer is still a 20-byte address, taken from the Keccak-256 hash of a type byte and its public key, following EIP-7932 and EIP-8051 [91, 92]. An owner adds one by updating the account's policy, and the account keeps its address and balances. Leaving the old key out of the policy retires it, and a retired key then fails at block admission, firm quotes, AMM `permit`, ERC-1271 checks and `ecrecover`, in line with the intent of EIP-8151 [93]. Each signature names its kind, so a later scheme such as FN-DSA only needs a new kind and a verifier. An account can also use ML-DSA from the start, because a native transaction can carry an ML-DSA signature in place of a secp256k1 one, with its sender derived from the key, so that account needs no secp256k1 key and no relayer. This costs more, since a plain transfer needs 270,280 gas (with illustrative prices) against 21,000 for secp256k1, which means a block of 100,000,000 gas holds about 370 of them at about 3.8 KB each, and these transactions can't be sponsored. They are a native format that standard Ethereum wallets can't sign, although Joltzu Wallet can (Section 8). An upgraded account's approval carries about 3.7 KB, paid for as call input, and it reaches the chain in a transaction from any funded account calling its account, including one that signs with ML-DSA itself. Services whose signers belong to users or issuers, such as grants, recovery, vouchers and quotes, use secp256k1. A policy doesn't move anyone by itself, though, so an exchange that signs standard Ethereum transactions stays on secp256k1 until it chooses to upgrade.

The VRF in JSN will be quantum safe as well, even though it sits outside the protocol. Elliptic-curve proofs won't do, because the public key appears in every fulfilment, so a machine that can break secp256k1 could recover the proving key and would then know each output as soon as a request is committed, which hands the advance knowledge we describe for the operator's prover (Section 17.15) to whoever owns that machine. We can't simply switch it to ML-DSA, for the same reason block randomness doesn't use it (Section 2.2), since a prover could keep signing until the output suited it, and RFC 9381 only defines RSA and elliptic-curve constructions, neither of which survives a quantum computer. We know of no standardized post-quantum VRF. The hash-based reveals that consensus uses are one candidate, because a committed Merkle root fixes exactly one value for each position and a contract can check it with a few dozen hashes, although each position could only serve one request and the prover would need to commit to a fresh root whenever a tree runs out. None of this changes withholding, because the prover still sees each output first and can decline to deliver it.

The BN254, BLS12-381 and KZG precompiles exist only for EVM compatibility and won't be safe for authorization once quantum machines exist.

As of 31 October 2026 the Ethereum Foundation's post-quantum team estimates that Ethereum's protocol upgrades could be completed by 2029, with no fixed date and full execution-layer migration taking years longer [94], and Arc has published its own roadmap [95]. EVM compatibility depends only on the execution layer, so we can follow Ethereum's account work without matching its validator signatures. None of this makes Joltzu fully quantum-resistant. Any account still relying on secp256k1 is open to a machine that can break it, and new keys cannot protect ciphertext that is already public.

17. Attacks

Section 7 tests the design against past thefts. Here we go through the assumptions that Joltzu's security rests on and how the design responds to attack, from consensus and the peer network to services, software, users and governance. Where the design leaves a risk open we say so.

17.1. Security assumptions

For ledger safety we assume that Byzantine validators hold less than a third of the global committee's assigned weight, and as long as that is true no height can finalize two blocks, whatever any group does (Appendix A). Liveness is more fragile. A certificate needs more than two-thirds of its committee's total assigned weight (offline members included), so a third or more of a group's weight can stop that group. Because the default inclusion policy requires a batch from every group in each block, one stalled group stalls the ledger under that policy. Clocks and message delays can affect liveness but not agreement. Joltzu also assumes that its signature and hash schemes are secure and that clients start from a checkpoint they trust (Section 17.3). Consensus does not depend on JSN at all. This analysis doesn't settle whether the adversary may corrupt validators after committees are chosen, and Section 17.5 explains why that choice matters.

17.2. Double spends and conflicting state

A double spend is when a user tries to spend the same funds more than once. For instance, Alice might try to spend the same 100 JOLT twice. Both transactions have to go through the group assigned to her account, which sees both and rejects the nonce conflict before anything is proposed. Execution in the global order then checks balances and shared contract state, so only the spending that the resulting state supports can settle.

17.3. Long-range attacks and weak subjectivity

A long-range attack is when validators from an old epoch come back and try to rewrite history. They cannot simply replay an old committee's certificates into a later epoch, because every commit certificate is checked against the committee that governs its height and a new committee is only accepted through the certified block that announced it. This does not stop validators who once held more than two-thirds of a committee's weight from withdrawing their stake and later signing an alternative history that branches inside their epoch. Their conflicting votes stay valid evidence indefinitely, but stake already withdrawn escapes the penalty.

This is why Joltzu relies on weak subjectivity, which means a client is only protected if it starts from a recent checkpoint that it got independently, and a checkpoint handed over by a peer doesn't count. Light clients pin their trust root. A new validator replays from genesis, so without one it accepts the first certified history it is served. A trusted checkpoint must be no older than the unbonding period (seven days in the illustrative parameters).

17.4. Equivocation and validator key compromise

Equivocation is when a validator signs conflicting votes. Signing keys live in a separate signer process, and each validator records every vote durably before asking for a signature, so a crash and restart cannot make it contradict itself. A validator that does sign conflicting votes can be reported and slashed (Section 2.6). Two ML-DSA signatures over the same vote can differ, so detection compares the signed statements and never the signature bytes. That durable record only helps with crashes. An operator who restores a stale backup or runs a second signer with the same key can still produce conflicting votes. Key custody and operations have to rule out both.

17.5. Group capture and randomness bias

Groups parallelize consensus, but they put fewer validators behind each certificate. The global committee can consist of the validators with the most bonded stake, up to a size limit, and the epoch seed can shuffle it into groups whose member counts differ by at most one. The shuffle balances member counts and ignores weight, so one heavy validator can hold a third or more of a group's weight while holding far less of the committee's, and any random split can leave some group with a larger Byzantine share than the committee as a whole. This means capturing a small group costs less than capturing the global committee.

A captured group cannot fork the ledger or insert an invalid batch, because global validators re-verify every batch (Appendix A). It can still withhold certificates, which stalls the ledger under the default policy, or censor every sender in its account range until membership changes. Assignments stay public for a whole epoch, which leaves time to bribe or attack a group, and withholding only costs slot rewards. The seed itself can be biased by a proposer that withholds its block (Section 2.2). Security at this layer depends on group sizes, weight-aware assignment, the corruption model and the resulting bound on the probability that any group exceeds its fault threshold.

17.6. Network partitions

The CAP theorem states that during a network partition you can only have consistency or availability. When a quorum is unavailable Joltzu chooses consistency over liveness. Thresholds count total assigned weight (not just the online stake), so separate partitions cannot each reach a quorum and finalize conflicting blocks. Dependencies between groups also affect availability, because under the default inclusion policy one unavailable group stalls the ledger (Section 17.1). Recovery has to handle that dependency explicitly.

17.7. Follow the leader attacks

This attack is when the leader is specifically targeted with some form of (D)DOS and then, when the leader rotates, the attackers move to the next leader. The proposer schedule is public, which is how an attacker can go after leaders as their slots approach. Joltzu reduces the impact of this because rotation and concurrent group consensus spread the pressure and epoch randomness keeps later schedules unpredictable, although disrupting one group can still delay the shared ledger. Recovery has to deal with targeted outages directly, without asking users to hold accounts in several ranges.

17.8. Censorship

A group's scheduled leader chooses which pending transactions go into its batch, and members only vote on whether a batch is valid and do not check whether it is complete, which is how a leader can omit a transaction. Rotation limits the damage, because the leader changes at every height and round and an omitted transaction stays in honest members' pools until the next honest leader includes it. A captured group can still exclude its senders until membership changes (Section 17.5), and Joltzu has no forced-inclusion mechanism.

Under the default policy the global proposer cannot drop a group's batch. Where a policy does allow omission, honest voters reject a first-round selection that leaves out a batch

they have already verified, but later rounds accept omissions so that a slow group cannot block the chain, which is why omission depends on an authenticated inclusion cutoff that every validator can check. The protocol also has to limit how long a fault-tolerant group can delay a valid transaction, because services with deadlines, like liquidations, quote expiries and challenge windows, are sized against that limit.

17.9. Front-running and transaction ordering

Neither a group leader nor the global proposer gets to choose the execution order, because validators recompute the deterministic order before voting (Section 2.4). That order follows fees and ignores arrival time, so anyone who sees a pending transaction can still pay to execute just before it and place another just after it, which is the pattern behind front-running and sandwich attacks. Pending transactions are public because peers relay them in plaintext.

This means users are protected by the limits they sign and cannot count on execution order. An orderbook order will not fill beyond its limit price, and a firm quote can name its permitted taker (Section 12.4). A swap is protected by the minimum output that its router or calling contract enforces, because native AMM pairs have no built-in router. Joltzu claims no resistance to ordering-based extraction.

17.10. Timestamp manipulation

Holds, bonded pauses, time locks, voucher deadlines, recurring payments, escrow and inheritance windows, unbonding and oracle freshness all read the consensus block timestamp instead of a validator's own clock. A proposer cannot write whatever time it likes into a block, because each block sits in a 500 ms slot counted from the genesis time, each slot has to be later than the one before it, and an honest validator will not prepare a proposal until its own clock has reached that slot. A proposal dated up to 15 seconds ahead (an illustrative value) is held until the validator's clock reaches that slot, and anything dated further ahead is dropped. This means a proposer cannot move time backwards or push it past the clocks of the honest validators in its quorum.

A proposer can hold time back though, because nothing bounds how early a first-round proposal may be dated, and that lets a faulty proposer delay deadlines until the next honest block. Services also resist repeated or delayed triggers. For instance, a repeated notice does not restart its window and missed recurring periods are skipped instead of being paid in a burst. Because deadlines carry no inclusion guarantee, users and services need margins. A validator whose clock falls behind a slot it has signed stops signing until it catches up, so clock faults only affect liveness.

17.11. Eclipse and peer attacks

In an eclipse attack a node gets surrounded by an attacker's peers. Validator links are mutually authenticated TLS 1.3 connections, and each side checks the other's certificate, issued by a single network certificate authority, against an explicit allowlist that validator operators configure. These links pin the hybrid `X25519MLKEM768` key exchange, so a classical-only negotiation fails, and their certificates use ML-DSA-65, since size matters little in a handshake. Since an unlisted peer is refused before any of its messages is queued, a Sybil attacker gains nothing by creating identities. A transport identity carries no voting authority either, as every vote, proposal and certificate is still checked against committee keys, so a peer that isolates a node can delay it but not deceive it.

A node that is catching up only replays complete proposals with commit certificates, re-executing each block and comparing the result with the certified commitments. It treats an empty reply as an unverified claim and rotates requests across its configured peers. Before permissionless operation Joltzu needs to bind transport identities to registered validator keys and replace or harden that certificate authority, and nodes also need to detect possible eclipse by comparing heads across independent peers.

17.12. Flooding and denial of service

A malicious bot farm may send legitimate transactions as quickly as possible to overload the network. This isn't without cost, because execution reserves a transaction's full maximum fee before running it, so a sender cannot execute more work than its balance pays for. Each node limits its pending pool both overall and per sender, although the pool does not check balances.

Every input is size-checked before it is buffered. Calldata, access lists, RPC requests and peer frames all have fixed caps, and an oversized frame closes the connection. When a peer exceeds its share of a node's inbound queue, only that peer's frames get dropped. Validators accept no public connections, so public traffic reaches them through non-voting nodes that hold no signing keys, including those behind JSN's authenticated gateways with per-client budgets (Section 13.2). The only fee floor is a nonzero gas price. If enough voting weight does go down then the partition rules kick in.

17.13. State growth

Contract storage pays the standard EVM storage gas, native services charge fixed gas before doing any work and every committed state record has a bounded key and value size. But Joltzu has no state rent or expiry, and some native records, such as challenge indices and tombstones, are kept permanently, which means state grows with use.

17.14. Replay and signature misuse

An attacker can also try to use a signed message again, later or somewhere else. Every native signed format begins with its own tag and version and includes the chain ID, and Ethereum transactions have to carry a chain ID, so a signature made for one purpose or chain verifies for no other.

Group committees also sign under a domain derived from their assignment. Transactions and account approvals consume separate nonces, and reentrancy cannot replay an approval because its nonce is spent before any of its calls run. Nobody can alter a signature into a second valid transaction either, since every secp256k1 signature must use the low-s form and Ethereum transactions must be canonically encoded.

Not every signature is single-use. An account-message proof can be checked repeatedly until it expires, and native ERC-1271 answers, like ERC-1271 itself, carry no expiry or nonce. This means applications have to include their own domain and a one-time nonce in the hash they ask an account to sign.

17.15. Compromised service operator

Since a single operator runs all of JSN, we have to assume that operator can go down or get hacked. Consensus carries on regardless, but what can go wrong is different for each service. For instance, RPC endpoints answer from the operator's own nodes, so reads can be false or withheld. Applications that cannot trust them verify state and receipt proofs or run their own node, and a dropped transaction can be resubmitted through any node without settling twice.

As host, the operator could serve tampered frontend code at a registered origin, which is invisible to origin binding. Protection falls to the wallet's decoding and simulation, to code eligibility and to grants that cannot widen without fresh consent. Hashes of approved artifacts also need to be published for wallets to compare. The VRF prover can compute each output as soon as a request is committed and could withhold one it dislikes. Applications need to treat an undelivered request as void instead of requesting again. Only registered workers execute keeper calls, so the operator can delay them until they expire, while native recurring payments remain open to anyone. SigNet peers cannot alter signed messages but can withhold them, and vouchers redeem without SigNet. A compromised co-signer could refuse to sign, which the everyday and master keys get around together, or sign whatever it is sent, which leaves the account as safe as its everyday key on its own was before it enabled co-signing. The co-signer also sees the IP addresses and devices that requests come from, so it has to publish how long it keeps them, and owners who run their own co-signer avoid all of this.

While one operator runs every AI worker, a committee's agreement only reflects that operator's judgement, so automated holds should stay disabled until independent operators are admitted. The operator also sees which addresses and contracts clients query, and it has to publish how long it keeps that metadata.

17.16. Software supply chain

Bybit's signers approved a theft through a tampered interface (Section 7), which is why we assume that any client software can be compromised. Validators enforce network eligibility, account policies, outflow budgets and locks whichever software signed a transaction, which limits what a tampered wallet or frontend can do. Joltzu Wallet's approvals also bind the complete request. Where code eligibility is enforced, validators only accept runtime code whose hash the code-admission committee approved, so a compromised compiler or dependency cannot deploy unreviewed bytecode. The node's toolchain, compilers and dependencies are pinned by version and hash, and it runs the exact image it built. Pinning fixes what gets built but cannot show that a distributed binary came from reviewed source, which takes reproducible builds, signed releases, a reviewed update channel for the wallet and signed publication of SDK packages.

17.17. Address poisoning and impersonation

With address poisoning, attackers plant lookalike addresses in a victim's history, hoping that the victim later copies the wrong one into a payment. They also imitate websites to collect approvals. Joltzu Wallet keeps favourite contacts, resolves aliases to addresses and flags a changed destination (Section 8), and checks websites for lookalike names and against the Domain Registry (Section 8.3). Protocol rules have no way to tell a mistaken recipient from an intended one, so the wallet also has to warn before a first payment to an address that resembles a saved contact.

17.18. Governance and committee attacks

Committees exist so that someone can step in, and that power brings its own risks, because keys can be compromised, members can collude, repairs can be faulty and restrictions can be repeated. Splitting authority up limits what any one committee can do, and supermajority thresholds, recorded actions, expiries and review deadlines constrain the rest.

17.19. Application and native-service exploits

A service can look correct alone and still break when combined with another, which is why security review needs to test both. It has to cover code admission and upgrades, simultaneous fills and withdrawals, stale prices, asset-policy changes, KYC on both sides of a settlement and the propagation of holds. Account policies have to hold on every spending path too, including while the wallet is closed, and so do safeguard allowances, which is why the ledger checks them on every debit rather than at each service. Contract invariants help here as well, because they check the state a combination of services leaves behind rather than any one service's logic.

A relayer that supplies too little gas could burn a user's approval. Grant and recurring-payment services prevent this by checking that they can forward an exact amount of gas before consuming an authorization. Account plans can sign the least gas the plan has to receive, and an approval that leaves it at zero accepts whatever the relayer forwards, so a

relayer that supplies too little can make such a plan fail and spend its nonce. Pre-call hooks are capped at 100,000 gas and invariant checks at 200,000, both paid from the transaction that triggers them.

17.20. Credential, advertising and voucher abuse

A KYC-gated service relies on an issuer's word about a user, and false credentials, stale attestations and compromised issuers all threaten it. Ads and vouchers have different problems and also need testing against hostile content, duplicate claims, parallel collectors and reused budgets. It helps that a signed message on its own doesn't authorize a payment.

17.21. AI network and intent attacks

Joltzu's AI agents can get things wrong by themselves or be attacked, and security review needs to cover both. Correlated errors in generated code and tests, nondeterminism and false alerts are the first kind, while malicious frontends, worker collusion, prompt injection and poisoned telemetry are the second. Joltzu addresses these risks with request-bound evidence, independent verification and externally enforced permissions, and neither a good reputation nor a persuasive explanation can authorize spending.

18. Conclusion

We are building Joltzu because we think everyday users need a blockchain that protects them by default. Native accounts and their safeguards, contract invariants, outflow budgets, bonded pauses and separated authority make theft slower and recovery more likely, and they do this without halting the network. Native markets and payments give developers safe building blocks on one shared ledger, and sharded consensus delivers the throughput those applications need.

The two parts of Joltzu ask for different kinds of trust. The protocol is decentralized because validators decide the ledger. The Joltzu Service Network is distributed but run by one operator, although consensus never depends on it and validators recheck whatever it submits. Its claims have to be earned through independent review, published benchmarks and real application use.

Appendix A: Formal Specification of Sharded Consensus

Two TLA+ modules go with this paper, and between them they specify Joltzu's consensus. `HotStuff.tla` specifies the protocol that every slot runs, whether that is a group slot certifying a batch or a global slot finalizing a block, while `ShardedConsensus.tla` composes group slots and global slots into the ledger. Both modules follow the reference node's vote rules.

Model

The state of the model is the set of votes that honest validators have cast. A validator's round is the latest round in which it has voted, and the validator may move to any later round at any time, as it would after a timeout. Its lock is its latest commit vote. The model treats Byzantine validators as having signed every statement, which is why their weight counts towards every certificate. We do not model leaders, because an honest validator may vote for any value its rules allow, which covers every behaviour of an equivocating leader. Messages may be delayed arbitrarily, and the set of values that honest validators accept as valid only grows. Listing 5 shows the core of the slot protocol.

```
Certified(s, r, ph, v) ==
  LET voters == {p \in Honest \cap Committee(s) :
        [p |-> p, s |-> s, r |-> r, ph |-> ph,
         v |-> v] \in votes}
  IN  3 * (SumW(voters) + FaultyW[s])
        > 2 * TotalW[s]

Final(s) == {v \in Values(s) :
  \E r \in Rounds(s) : Certified(s, r, "commit", v)}

SafeToPrepare(p, s, r, v) ==
  LET l == Lock(p, s) IN
  IF l = NoLock THEN TRUE
  ELSE \/ l.v = v
       \/ \E r2 \in Rounds(s) : l.r < r2 /\ r2 < r
             /\ Certified(s, r2, "prepare", v)

Prepare(p, s, r, v) ==
  /\ MayVote(p, s, r, "prepare", v)
  /\ v \in ValidIn(s)
  /\ SafeToPrepare(p, s, r, v)
  /\ Cast(p, s, r, "prepare", v)

PreCommit(p, s, r, v) ==
  /\ MayVote(p, s, r, "precommit", v)
  /\ Certified(s, r, "prepare", v)
  /\ LET l == Lock(p, s) IN
     IF l = NoLock THEN TRUE ELSE l.v = v \/ l.r < r
  /\ Cast(p, s, r, "precommit", v)

Commit(p, s, r, v) ==
  /\ MayVote(p, s, r, "commit", v)
  /\ Certified(s, r, "precommit", v)
  /\ Cast(p, s, r, "commit", v)
```

Listing 5: *The slot protocol.*

`MayVote` requires a validator to be on the committee and voting in a round no earlier than its latest vote. It also forbids a second vote in the same phase and round, and it limits each validator to one value per round across all phases. As in the reference node, a certificate counts absent members towards the total weight.

In the composition, the global slot at each height runs the same rules over manifests, and a manifest assigns each group at most one batch. A global validator only accepts a manifest when every included batch is valid and was finalized by its own group in the same slot (Listing 6). Because `HotStuff.tla` shows that the slot protocol implements an atomic consensus object, the composition represents each group slot by that object. In that object a fault-tolerant group finalizes at most one batch, which is valid, while a

group beyond its fault bound may finalize anything, including several conflicting or invalid batches.

```
AcceptsManifest(h, m) ==
  \A g \in Groups : m[g] = Omit \/
    (ValidBatch(m[g]) /\ m[g] \in groupFinal[g][h])

GroupFinalize(g, h, b) ==
  /\ Started(h) /\ b \notin groupFinal[g][h]
  /\ GroupTolerant(g) =>
       (groupFinal[g][h] = {} /\ ValidBatch(b))
  /\ groupFinal' =
       [groupFinal EXCEPT ![g][h] = @ \cup {b}]
  /\ UNCHANGED votes

LedgerAgreement == \A h \in Heights :
  Cardinality(Global!Final(h)) <= 1

LedgerValidity == \A h \in Heights :
  \A m \in Global!Final(h), g \in Groups :
    m[g] # Omit => ValidBatch(m[g])
                   /\ m[g] \in groupFinal[g][h]
```

Listing 6: *Composition and ledger properties.*

Safety theorems

Consider a fault-tolerant committee, by which we mean one with total weight W and Byzantine weight $F < W/3$. Since a certificate needs honest voters H with $w(H) + F > 2W/3$, every certificate includes honest weight greater than $W/3$.

Lemma 1. The honest voters H_1 and H_2 of any two certificates share a validator. *Proof.* $w(H_1) + w(H_2) > 4W/3 - 2F$, while $w(H_1 \cup H_2) \leq W - F$, so $w(H_1 \cap H_2) > W/3 - F > 0$.

Lemma 2. Two certificates of the same phase and round are for the same value. *Proof.* By Lemma 1 an honest validator voted in both, and it votes for one value per round.

Lemma 3. If v has a commit certificate in round r, no value $w \neq v$ has a prepare certificate in any round $r' > r$. *Proof.* Suppose otherwise and take the least such r'. Consider the honest votes that first formed the prepare certificate for w in round r'. By Lemma 1 one of them was cast by an honest validator p that voted commit for v in round r. That commit vote came before p's prepare vote, because a validator never votes in a round earlier than its latest vote. Also, p had not voted commit in round r', since no pre-commit certificate for w in round r' yet existed and p votes for one value per round. So when p voted prepare, its lock was (r_l, v_l) with $r \leq r_l < r'$. If $r_l > r$, the lock rests on a pre-commit certificate, and so also on a prepare certificate, for v_l in round r_l, which gives $v_l = v$ by the minimality of r'. Otherwise $v_l = v$ directly. As $w \neq v_l$, the prepare vote needed a prepare certificate for w in some round between r_l and r', which contradicts the minimality of r'.

Theorem 1 (slot safety). A fault-tolerant slot finalizes at most one value. An honest validator that voted prepare on that value accepted it as valid and stored it. *Proof.* Let v and w have commit certificates in rounds $r \leq r'$. If $r = r'$, Lemma 2 gives $v = w$. If $r < r'$, an honest voter of the commit certificate for w saw a pre-commit certificate for w in round r'. An honest voter of that certificate saw a prepare certificate for w in round r', so Lemma 3 gives $w = v$. The same chain from a commit certificate leads to an honest prepare voter, which validated and stored the value. The set of valid values only grows, so the value remains valid.

Theorem 2 (ledger safety). If the global committee is fault tolerant then, whatever any group does: (i) each height finalizes at most one block; (ii) every batch in a finalized block is valid and was finalized by its own group in the same slot, and a block holds at most one batch per group slot; and (iii) every finalized block, including its batches, is stored by an honest validator. *Proof.* Parts (i) and (iii) are Theorem 1 for the global slot. For (ii), Theorem 1 gives an honest validator that accepted the block. That validator accepts a block only after checking every batch's validity and its group's commit certificate, and neither can later become false. Finally, a manifest assigns each group at most one batch.

This means a group beyond its fault bound can equivocate or certify an invalid batch, but it cannot fork the ledger or place an invalid batch in it. At worst its slot is omitted or, under the default inclusion policy, the ledger stalls (Section 17.1).

Model checking

We ran the TLC model checker over the configurations in Table 3, where it explored every reachable state and found no violation of any property. Each configuration uses four validators of equal weight, one of them Byzantine. The slot configuration checks `Agreement`, `Validity`, `Available` and the temporal property `AtomicDecision`, which states that a single valid value becomes final in one step and never changes. The ledger configurations check `LedgerAgreement`, `LedgerValidity`, `LedgerAvailable` and `HeightsInOrder`.

Model	Scope	States
Slot protocol	One slot, rounds 0–2, two values	1,892,899
Two groups	One height, global rounds 0–1	197,650
Two heights	One group, one global round per height	25,317
Byzantine group	Two groups, one Byzantine; one height, global rounds 0–1	1,036,320

Table 3: *Exhaustive TLC runs: distinct states explored, after symmetry reduction, with no violation found.*

We also made sure these checks are not vacuous. TLC confirms that blocks do become final, that a block can include batches from both groups and that a second height can finalize. If we remove the lock rule from `SafeToPrepare`, or lower the quorum to half the total weight, TLC produces a slot that finalizes two values, and if we remove the global validators' re-validation of batches, the Byzantine group can place an invalid batch in the ledger.

Scope

The theorems cover safety for any committee size and number of rounds, while model checking confirms them for the bounded configurations above. The specification doesn't cover liveness under partial synchrony, epoch and committee transitions, RANDJAO or the deterministic ordering rule.

References

[1] Binance. Initial Findings on the October 2026 Security Incident. Post on X, https://x.com/binance/status/2115993361517891592, 2026. Accessed 30 October 2026.

[2] Sygnia. Sygnia's Investigation into the Bybit Hack: What We Know So Far. https://www.sygnia.co/blog/sygnia-investigation-bybit-hack/, 2025. Accessed 8 October 2026.

[3] Scam Sniffer. Scam Sniffer 2024: Web3 Phishing Attacks – Wallet Drainers Drain $494 Million. https://drops.scamsniffer.io/scam-sniffer-2024-web3-phishing-attacks-wallet-drainers-drain-494-million/, 2025. Accessed 11 October 2026.

[4] Ethereum.org. Ethereum Virtual Machine (EVM). https://ethereum.org/en/developers/docs/evm/.

[5] National Institute of Standards and Technology. FIPS 204: Module-Lattice-Based Digital Signature Standard. https://csrc.nist.gov/pubs/fips/204/final, 2024. Accessed 31 October 2026.

[6] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT Consensus in the Lens of Blockchain. arXiv preprint arXiv:1803.05069v6, 2019. Full version; conference version published in PODC 2019.

[7] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[8] Solidity Documentation. Security considerations. https://docs.soliditylang.org/en/latest/security-considerations.html.

[9] Francisco Giordano, Matt Condon, Philippe Castonguay, Amxx, Julien Niset, and Stefan George. ERC-1271: Standard Signature Validation Method for Contracts. https://eips.ethereum.org/EIPS/eip-1271, 2018.

[10] Stellar Development Foundation. Signatures and multisig. https://developers.stellar.org/docs/learn/fundamentals/transactions/signatures-multisig. Accessed 13 September 2026.

[11] Circle. Arc: Stablecoin native model. https://docs.arc.io/arc/concepts/stablecoin-native-model. Accessed 13 September 2026.

[12] RocksDB. Getting Started. https://rocksdb.org/docs/getting-started.html.

[13] Cap'n Proto. Encoding spec. https://capnproto.org/encoding.html.

[14] Sui Foundation. Object-based local fee markets. https://docs.sui.io/develop/transaction-payment/local-fee-markets. Accessed 13 September 2026.

[15] PyTorch. Reproducibility. https://docs.pytorch.org/docs/stable/notes/randomness.html.

[16] Lorenz Breidenbach, Christian Cachin, Alex Coventry, Yan Ji, Kostis Karantias, Philipp Schindler, Chrysoula Stathakopoulou, and Alexandru Topliceanu. Chainlink Offchain Reporting Protocol 3.0. Protocol report, Chainlink Labs, 18 May 2025.

[17] KD Conway, Cathie So, Xiaohang Yu, and Kartin Wong. opML: Optimistic Machine Learning on Blockchain. arXiv preprint arXiv:2401.17555v2, 2024.

[18] Ethereum Improvement Proposals. ERC-8004: Trustless agents. https://eips.ethereum.org/EIPS/eip-8004. Draft; created 13 August 2025. Accessed 13 September 2026.

[19] Apostol Vassilev, Alina Oprea, Alie Fordyce, and Hyrum Anderson. Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations. NIST AI 100-2e2023, National Institute of Standards and Technology, January 2024.

[20] Chainalysis. North Korea Drives Record $2 Billion Crypto Theft Year, Pushing All-Time Total to $6.75 Billion. https://www.chainalysis.com/blog/crypto-hacking-stolen-funds-2026/, 2025. Accessed 29 September 2026.

[21] TRM Labs. 2026 Crypto Crime Report. https://www.trmlabs.com/reports-and-whitepapers/2026-crypto-crime-report, 2026. Accessed 10 October 2026.

[22] TRM Labs. H1 2026 Crypto Hacks Reach Record High as Losses Fall Below USD 1 Billion. https://www.trmlabs.com/resources/blog/h1-2026-crypto-hacks-reach-record-high-as-losses-fall-below-usd-1-billion, 2026. Accessed 27 October 2026.

[23] Elliptic. The Largest Theft in History: North Korea Likely Behind the $6.4 Billion Binance Hack. https://www.elliptic.co/insights/binance-largest-hack-in-history/, 2026. Accessed 30 October 2026.

[24] rekt.news. Rekt Leaderboard. https://rekt.news/leaderboard, 2026. Accessed 20 October 2026.

[25] rekt.news. Ronin Network - REKT. https://rekt.news/ronin-rekt, 2022. Accessed 29 October 2026.

[26] Elliptic. The Poly Network Hack: $600 Million in Crypto Stolen and Returned in 24 Hours. https://www.elliptic.co/blog/the-poly-network-hack-600-million-in-crypto-stolen-and-returned-in-24-hours, 2021. Accessed 22 October 2026.

[27] rekt.news. BNB Bridge - REKT. https://rekt.news/bnb-bridge-rekt, 2022. Accessed 2 November 2026.

[28] TRM Labs. Bitget Loses USD 351.6 Million in Hot Wallet Breach in Likely North Korea Attack. https://www.trmlabs.com/resources/blog/bitget-loses-usd-3516-million-in-hot-wallet-breach-in-likely-north-korea-attack, 2026. Accessed 2 November 2026.

[29] rekt.news. Wormhole - REKT. https://rekt.news/wormhole-rekt, 2022. Accessed 27 October 2026.

[30] TRM Labs. 2026's Biggest Hack To Date: Attackers Drained USD 319 Million in Bitcoin From Liquid Network, Then Returned 85% of Funds. https://www.trmlabs.com/resources/blog/2026s-biggest-hack-to-date-attackers-drained-usd-319-million-in-bitcoin-from-liquid-network-then-returned-85-of-funds, 2026. Accessed 9 October 2026.

[31] LayerZero Labs. LayerZero Labs KelpDAO Incident Report. https://layerzero.network/blog/layerzero-labs-kelpdao-incident-report, 2026. Accessed 5 October 2026.

[32] TRM Labs. North Korean Hackers Attack Drift Protocol In USD 285 Million Heist. https://www.trmlabs.com/resources/blog/north-korean-hackers-attack-drift-protocol-in-285-million-heist, 2026. Accessed 26 October 2026.

[33] Elliptic. $235 million lost by WazirX in North Korea-linked breach. https://www.elliptic.co/blog/235-million-lost-by-wazirx-in-north-korea-linked-breach, 2024. Accessed 15 October 2026.

[34] Cyfrin. Inside The $223M Cetus Exploit: Root Cause And Impact Analysis. https://www.cyfrin.io/blog/inside-the-223m-cetus-exploit-root-cause-and-impact-analysis, 2025. Accessed 12 October 2026.

[35] rekt.news. Euler Finance - REKT. https://rekt.news/euler-rekt, 2023. Accessed 26 October 2026.

[36] TRM Labs. Key Suspect in $190M Nomad Bridge Exploit Extradited to the United States. https://www.trmlabs.com/resources/blog/key-suspect-in-190m-nomad-bridge-exploit-extradited-to-the-united-states, 2025. Accessed 2 October 2026.

[37] Beanstalk Farms. Beanstalk Governance Exploit. https://bean.money/blog/beanstalk-governance-exploit, 2022. Accessed 29 September 2026.

[38] Trail of Bits. Balancer hack analysis and guidance for the DeFi ecosystem. https://blog.trailofbits.com/2025/11/07/balancer-hack-analysis-and-guidance-for-the-defi-ecosystem/, 2025. Accessed 29 September 2026.

[39] rekt.news. Multichain - REKT 2. https://rekt.news/multichain-rekt2, 2023. Accessed 12 October 2026.

[40] rekt.news. Badger - REKT. https://rekt.news/badger-rekt, 2021. Accessed 29 October 2026.

[41] U.S. Commodity Futures Trading Commission. CFTC Charges Avraham Eisenberg with Manipulative and Deceptive Scheme to Misappropriate Over $110 million from Mango Markets, a Digital Asset Exchange. https://www.cftc.gov/PressRoom/PressReleases/8647-23, 2023. Accessed 29 September 2026.

[42] Elliptic. The $100 million Horizon hack: following the trail through Tornado Cash to North Korea. https://www.elliptic.co/blog/analysis/the-100-million-horizon-hack-following-the-trail-through-tornado-cash-to-north-korea, 2022. Accessed 15 October 2026.

[43] rekt.news. Orbit Bridge - REKT. https://rekt.news/orbit-bridge-rekt, 2024. Accessed 22 October 2026.

[44] Vyper Team. Vyper Nonreentrancy Lock Vulnerability Technical Post-Mortem Report. https://hackmd.io/@vyperlang/HJUgNMhs2, 2023. Accessed 6 October 2026.

[45] LlamaRisk. Curve Pool Reentrancy Exploit Postmortem July 30th, 2023. https://hackmd.io/@LlamaRisk/BJzSKHNjn, 2023. Accessed 27 October 2026.

[46] TRM Labs. North Korea Steals USD 6.36 Billion From Binance in the Largest Hack in History. https://www.trmlabs.com/resources/blog/north-korea-steals-usd-6-36-billion-from-binance-in-the-largest-hack-in-history, 2026. Accessed 30 October 2026.

[47] Federal Bureau of Investigation. North Korea Responsible for $1.5 Billion Bybit Hack. https://www.ic3.gov/psa/2025/psa250226, 2025. Public Service Announcement I-022625-PSA. Accessed 23 October 2026.

[48] TRM Labs. Bybit Hack Update: North Korea Moves to Next Stage of Laundering. https://www.trmlabs.com/resources/blog/bybit-hack-update-north-korea-moves-to-next-stage-of-laundering, 2025. Accessed 19 October 2026.

[49] Cointelegraph. Bybit CEO: 20% of $1.4B stolen funds 'gone dark'. https://cointelegraph.com/news/bybit-hack-280m-untraceable-1b-trackable-thorchain-okx-exch, 2025. Accessed 16 October 2026.

[50] Chainalysis. 2022 Biggest Year Ever For Crypto Hacking. https://www.chainalysis.com/blog/2022-biggest-year-ever-for-crypto-hacking/, 2023. Accessed 2 November 2026.

[51] BNB Chain. BNB Chain Ecosystem Update. https://www.bnbchain.org/en/blog/bnb-chain-ecosystem-update, 2022. Accessed 28 October 2026.

[52] Sui Foundation. Response to the Cetus Incident – Onchain Community Vote. https://www.sui.io/blog/cetus-incident-response-onchain-community-vote, 2025. Accessed 29 September 2026.

[53] Arbitrum Security Council. Security Council Emergency Action – 21/04/2026. https://forum.arbitrum.foundation/t/security-council-emergency-action-21-04-2026/30803, 2026. Accessed 22 October 2026.

[54] The Block. Cronos says $9.2 million remains unrecovered after Tectonic exploit, chain rollback. https://www.theblock.co/news/ecosystems/2026-09-08-cronos-post-mortem-413724, 2026. Accessed 29 September 2026.

[55] Winnie Xiao, Cole Killian, Henry Sleight, Alan Chan, Nicholas Carlini, and Alwin Peng. AI agents find $4.6M in blockchain smart contract exploits. Anthropic, https://www.anthropic.com/research/smart-contracts, 1 December 2025. Accessed 12 October 2026.

[56] Newton Cheng, Keane Lucas, Winnie Xiao, Nicholas Carlini, and Milad Nasr. Measuring LLMs' ability to develop exploits. Anthropic, https://www.anthropic.com/research/exploit-evals, 22 May 2026. Accessed 6 October 2026.

[57] Arthur Gervais and Liyi Zhou. AI Agent Smart Contract Exploit Generation. arXiv:2507.05558, https://arxiv.org/abs/2507.05558, 2025. Accessed 16 October 2026.

[58] Richard Fang, Rohan Bindu, Akul Gupta, and Daniel Kang. LLM Agents can Autonomously Exploit One-day Vulnerabilities. arXiv:2404.08144, https://arxiv.org/abs/2404.08144, 2024. Accessed 26 October 2026.

[59] Chainalysis. The Hidden Code Problem: How Unverified Smart Contracts Are Becoming a Preferred Target for Attackers. https://www.chainalysis.com/blog/attackers-exploiting-unverified-smart-contracts/, 2026. Accessed 4 October 2026.

[60] Anthropic. Project Glasswing: An initial update. https://www.anthropic.com/research/glasswing-initial-update, 22 May 2026. Accessed 11 October 2026.

[61] DL News. Truebit hit by $26m exploit as attackers increasingly target older DeFi protocols. https://www.dlnews.com/articles/defi/truebit-hit-by-exploit-as-attackers-increasingly-target-older-defi-protocols/, 2026. Accessed 2 November 2026.

[62] Ethereum Improvement Proposals. ERC-7715: Request permissions from wallets. https://eips.ethereum.org/EIPS/eip-7715, 2024. Draft; created 24 May 2024.

[63] Unicode Consortium. Unicode Security Mechanisms. Unicode Technical Standard 39, 2026. Revision 34, 27 August 2026.

[64] Donald E. Eastlake 3rd. Domain Name System (DNS) Case Insensitivity Clarification. RFC 4343, IETF, 2006.

[65] Murray Kucherawy and Elizabeth Zwicky. Domain-based Message Authentication, Reporting, and Conformance (DMARC). RFC 7489, IETF, 2015.

[66] Vitalik Buterin, Yoav Weiss, Dror Tirosh, Shahaf Nacson, Alex Forshtat, Kristof Gazso, and Tjaden Hess. ERC-4337: Account abstraction using alt mempool. https://eips.ethereum.org/EIPS/eip-4337, 2021.

[67] ENS. Address lookup. https://docs.ens.domains/web/resolution/.

[68] Unstoppable Domains. Resolution: Blockchain domain name resolution library. https://github.com/unstoppabledomains/resolution.

[69] Brave Software. What is brave rewards? https://brave.com/brave-rewards/. Accessed 14 September 2026.

[70] Circle. Arc: Stable fee design. https://docs.arc.io/arc/concepts/stable-fee-design. Testnet design. Accessed 13 September 2026.

[71] Sui Foundation. Ethereum to sui: Programmable transaction blocks. https://docs.sui.io/getting-started/sui-for-ethereum. Accessed 13 September 2026.

[72] Stripe. Place a hold on a payment method. https://docs.stripe.com/payments/place-a-hold-on-a-payment-method. Accessed 19 September 2026.

[73] Stripe. How disputes work. https://docs.stripe.com/disputes/how-disputes-work. Accessed 19 September 2026.

[74] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Lightning Network, January 2016. Draft version 0.5.9.2, 14 January 2016.

[75] Remco Bloemen, Leonid Logvinov, and Jacob Evans. EIP-712: Typed Structured Data Hashing and Signing. Ethereum Improvement Proposals, 2017.

[76] World Wide Web Consortium. Verifiable credentials data model v2.0. https://www.w3.org/TR/vc-data-model-2.0/, 2025. W3C Recommendation, 15 May 2025.

[77] Fabian Vogelsteller and Vitalik Buterin. ERC-20: Token Standard. https://eips.ethereum.org/EIPS/eip-20, 2015.

[78] Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 Core. https://uniswap.org/whitepaper.pdf, March 2020.

[79] Circle. Arc: Contract addresses and settlement infrastructure. https://docs.arc.io/arc/references/contract-addresses. Testnet interfaces. Accessed 13 September 2026.

[80] Chainlink. Selecting quality data feeds. https://docs.chain.link/data-feeds/selecting-data-feeds.

[81] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Vcelak. Verifiable Random Functions (VRFs). RFC 9381, RFC Editor, 2023.

[82] David M'Raihi, Salah Machani, Mingliang Pei, and Johan Rydell. TOTP: Time-Based One-Time Password Algorithm. RFC 6238, RFC Editor, 2011.

[83] Trail of Bits. Slither: Static Analyzer for Solidity and Vyper. https://github.com/crytic/slither.

[84] Trail of Bits. Echidna: Ethereum Smart Contract Fuzzer. https://github.com/crytic/echidna.

[85] Solidity Documentation. SMTChecker and Formal Verification. https://docs.soliditylang.org/en/latest/smtchecker.html.

[86] Murugiah Souppaya, Karen Scarfone, and Donna Dodson. Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities. NIST SP 800-218, National Institute of Standards and Technology, February 2022.

[87] Ethereum Improvement Proposals. ERC-1167: Minimal Proxy Contract. https://eips.ethereum.org/EIPS/eip-1167, 2018.

[88] Ryan Babbush et al. Securing Elliptic Curve Cryptocurrencies against Quantum Vulnerabilities: Resource Estimates and Mitigations. https://arxiv.org/abs/2603.28846, 2026. Accessed 31 October 2026.

[89] leanEthereum. leanSig. https://github.com/leanEthereum/leanSig. Unaudited prototype. Accessed 31 October 2026.

[90] leanEthereum. leanMultisig. https://github.com/leanEthereum/leanMultisig. Accessed 31 October 2026.

[91] Ethereum Improvement Proposals. EIP-7932: Secondary signature algorithms. https://eips.ethereum.org/EIPS/eip-7932, 2025. Draft. Accessed 31 October 2026.

[92] Ethereum Improvement Proposals. EIP-8051: Precompile for ML-DSA signature verification. https://eips.ethereum.org/EIPS/eip-8051, 2025. Draft. Accessed 31 October 2026.

[93] Ethereum Improvement Proposals. EIP-8151: Account code restricted ecrecover. https://eips.ethereum.org/EIPS/eip-8151, 2026. Draft. Accessed 31 October 2026.

[94] Ethereum Foundation. Post-Quantum Ethereum. https://pq.ethereum.org/, 2026. Last updated 27 June 2026. Accessed 31 October 2026.

[95] Circle. Arc: Post-quantum security. https://docs.arc.io/arc/concepts/post-quantum-security. Roadmap. Accessed 13 September 2026.

About the Author

Sam Witch is a software developer and founder who has worked in the cryptocurrency space since 2017. A co-founder of Paintswap and Estfor Kingdom, he is the master mind behind the Paintswap NFT Marketplace and Launchpad, Estfor Kingdom, Paintswap VRF, Click Royale stress tester, Paintswap's AI Game Engine, the fNFT orderbook for Sonic's official airdrop, and now the Joltzu protocol. His work also includes contributions to Flying Tulip and several other NDA'd protocols. His public code portfolio includes recovering user funds in a documented token recovery and live recovery. His broader software experience spans CEX matching engines, 3D CAD/CAM manufacturing software tools, games, and game-engine plugins.