ISSUANCE CONSOLE · VERIFIABLE REGISTRY

Cachet

An open registry and browser mint studio
for Zcash Shielded Assets

A working paper: every claim in this document is measured, running at a public URL, or cited. Nothing is announced.

VERSION	DATE	LIVE AT
1.0	30 AUGUST 2026	CACHETZEC.COM
CHAIN	LICENSE	CONTACT
PUBLIC ZSA TESTNET	MIT	@CACHET_ZEC

TESTNET · ZERO MONETARY VALUE · NO MAINNET CLAIMS

ABSTRACT

What this is, and what it is not

Zcash Shielded Assets (ZSA) give Zcash native multi-asset support with a property no other production chain offers: issuance and supply are public and auditable, while holders, balances and transfers are cryptographically hidden. The protocol is specified (ZIPs 226 and 227), independently audited, and live on a dedicated public test network. What it lacked was tooling: at the time of writing there was no graphical way to issue an asset, no registry to browse what exists, and no way for a person to verify an asset's metadata without writing code.

Cachet fills that gap with three pieces, all running today at cachetzec.com: an issuance console, a verifiable registry that indexes every asset on the chain regardless of who issued it, and a browser mint studio in which seeds are generated, transactions are built, zero-knowledge proofs are computed and signatures are made entirely inside the visitor's browser, with the server reduced to a relay that can refuse but never alter.

This is a working paper, not a whitepaper in the promissory sense. Sections describe software that runs, with measurements taken on it, and the source will be published under the MIT license. Where something does not exist yet, the text says so plainly.

> A browser-minted proof of the whole pipeline: asset `3081f54e…cdb70e` (“Minted From A Browser”) was created from the live page on 30 August 2026, confirmed at height 520 of the public ZSA testnet, and is checkable against QEDIT's public node rather than any infrastructure of ours.

What already runs

Component	State	Where to check
Issuance console (mint, batch mint, transfer, burn)	Running	`cachetzec.com/console`
Registry: every asset on the chain, exact supplies	Running, 130+ assets indexed	`cachetzec.com`
In-browser metadata verification (SubtleCrypto)	Running	`any asset page`
Per-asset public history; transfers never shown	Running	`any asset page`
Issuer collections (assets grouped by issuance key)	Running	`cachetzec.com/issuers`
Permissionless, hash-verified description resolution	Running	`POST /api/v1/assets/{id}/description`
ZMD-1 descriptor recognition (third-party convention)	Running	`zecbit-genesis assets, resolved`
Operator moderation (availability-only denylist)	Running	`410 hidden-by-operator`
Browser mint studio (keys never leave the page; threaded, key-cached proving, 7.7x)	Running	`cachetzec.com/mint`
Signed-tx relay (mint relay, refuses or mines, never alters)	Running	`POST /api/v1/relay`
Read-only public deployment, throwaway seed only	Running	`this instance`

1 · THE GAP

A finished protocol with no front door

The ZSA protocol reached an unusual state in 2026: complete and idle. ZIP 227 specifies issuance, ZIP 226 specifies shielded transfer and burn, Least Authority audited the OrchardZSA implementation in January 2025, and since 2 August 2026 a dedicated public test network operated by QEDIT has enforced the rules from its genesis block. Activation on Zcash mainnet waits on a future network upgrade after the v6 format was deferred out of NU7, a governance timeline nobody controls.

Meanwhile the only way to issue an asset was a developer command-line workflow against pinned alpha forks of the core Rust libraries: clone the right revisions, set an unstable compiler flag, derive keys by hand, build and mine your own block. Wallet teams had nothing to test against; curious users had nothing to look at; the protocol's strongest argument, that it works, was invisible.

Cachet's bet is that the period before activation is exactly when the tooling should be built and hardened in public: on a chain with zero monetary value, mistakes are free, and when the protocol ships, the ecosystem starts with a registry, a mint studio and a browser signing stack instead of starting from zero. We call the posture “production-grade engineering around a testnet-grade product”.

The asset model, in one paragraph

An OrchardZSA asset's identity is derived, never assigned: the asset id is a function of the issuer's issuance validating key and the BLAKE2b-256 hash of an issuer-chosen description string (at most 512 bytes; only the hash is on chain). Issuance is a public event: issuer key, amounts and a finalize flag are visible to anyone. Once an asset is finalized, consensus refuses further units forever, from anyone, including the issuer. After issuance, notes move inside the Orchard shielded pool: holders, balances, amounts and counterparties are hidden, and an asset transfer is indistinguishable from any other shielded transaction. Public existence, private ownership. Everything Cachet does is an interpretation layer over exactly these facts.

2 · PRINCIPLES

Rules the software can be held to

Four rules carry every design decision. They are phrased so that violations are observable, and the deployment sections show how each is enforced structurally rather than by promise.

Rule	Meaning	Enforced by
The chain is the source of truth	Every fact served is recomputable from public chain data; the database holds only reconstructible derivations and the issuer's own description journal	deterministic indexer; drop-and-resync is routine (the index refolds in about a minute)
The registry can withhold, never lie	Metadata is content-addressed and its hash participates in the asset id; a registry can decline to serve bytes, it cannot substitute them	clients re-hash in the browser; moderation is availability-only (section 5)
No custody, ever	No accounts, no hosted wallets, no escrow phase, no ownership database; the public instance holds a throwaway seed and signs nothing	read-only mode; browser-side signing (section 6); wallet endpoint disabled in public deployments
Privacy is opposable	No telemetry, no analytics, no IP logging, strict CORS, no-referrer on outbound links, no third-party requests at runtime	docs/PRIVACY.md rules P1-P7; rate limiting keys live in memory only and are pruned each minute

A fifth, softer principle governs the text you are reading: measured claims only. Numbers in this paper come from benchmarks run on the shipped binaries and pages, and the paper prefers “we measured X on Y hardware” to “up to X”.

3 · THE REGISTRY

Indexing everyone, trusting no one

The indexer scans every block from height 1 and folds every issuance bundle and burn it finds, whoever the issuer is. For each asset it maintains: the asset id, the on-chain description hash, the issuer's issuance validating key (ZIP 227 canonical encoding), the exact supply (issued minus burned, computed, never sampled), the finalization state, and the public event history with heights and transaction ids. Transfers never appear anywhere, because they do not exist in public chain data; the interface says so explicitly on every asset page rather than leaving an ambiguous blank.

Sealed metadata: the v1 envelope

Assets minted through Cachet commit their metadata cryptographically. The display name and the SHA-256 of a content-addressed metadata bundle (long description, optional image) travel inside the on-chain description:

```
{"v":1,"name":"Zcon Ticket 2027","sha256":"<hex of the bundle bytes>"}
```

Because the description hash participates in the asset id, the bundle is bound to the asset forever: nobody, including the registry operator, can swap the name or image afterwards. Every asset page re-fetches the bundle, re-hashes it with the browser's SubtleCrypto and compares against the on-chain commitment, so integrity never depends on trusting the server. A separate creator signature is deliberately absent: the issuance bundle's own ZIP 227 signature already covers the description hash, and therefore, transitively, every byte of the metadata. A second signature only adds value for mutable metadata, which this format refuses to support.

Permissionless resolution, and foreign conventions

The chain stores only hashes, so the plaintext description of a foreign asset is unknown until someone provides it. Anyone can: the resolution endpoint accepts a candidate description and records it only if its personalized BLAKE2b-256 matches the on-chain commitment. No account, no signature, no trust; a matching preimage is definitionally correct, and the registry cannot be lied to. The endpoint stays open on read-only deployments because resolution is verification, not issuance.

The registry is also deliberately polyglot. Descriptions matching ZecBit's published ZMD-1 grammar are parsed (strictly: reject, never repair) and displayed under their canonical `slug #index` form. Every displayed name carries its provenance so the interface can render trust states honestly: envelope names are sealed into the asset id, zmd1 names are on-chain machine identifiers, free_text labels are issuer-chosen and unverified (shown dimmed and italic), and unresolved assets are identified by their id alone. A name is never shown without its provenance; this is the anti-phishing display rule ZIP 227 asks wallets to adopt.

Case study: what an independent registry is for

In August 2026 the ZecBit project published a proof-of-concept batch of ten supply-one finalized assets on the public testnet, presented as evidence of its issuance pipeline. Cachet indexed them automatically (they are simply assets on the chain), resolved their published descriptors, and displayed them, which also surfaced a fact the announcement did not mention: the batch was issued under the ecosystem's shared public demo key, the one shipped in QEDIT's reference tooling and held by everyone, so that particular batch is not cryptographically attributable to its author. No accusation follows from this; the point is structural. A registry that shows exactly what the chain says, no more and no less, is what keeps every issuer's story checkable, including ours.

4 · PROVENANCE

Issuers and collections

The chain makes exactly one provenance statement: these assets were minted under the same issuance key. Cachet exposes it as first-class data. Every asset response carries its issuer's validating key; a collections view groups the chain by issuer with asset counts, sealed counts and circulating supplies; and every asset page links to the issuer's full list. What the key does not say, the interface also states: nothing about who the issuer is in the real world. Real-world identity is an attestation problem that no chain can solve, and Cachet does not pretend to solve it either.

Because issuance is the only transparent moment in an asset's life, initial mint recipients appear in public issue notes. Cachet's own server wallet mints to itself (account 0) and distributes by shielded transfer afterwards, so the public record binds assets to the issuer, never to a customer address. Browser-minted assets behave the same way: the issue note pays the minter's own account-zero address, derived in the page.

5 · MODERATION

Availability, not truth

A public registry distributes bytes, so its operator carries responsibility for what it serves, and needs a lever that does not compromise the integrity story. The lever is an operator denylist with one rule: a registry can withhold, it can never lie.

```
cachet-server moderate list | hide-bundle <sha256> [reason] | unhide-bundle | hide-description
<asset_id> [reason] | unhide-description
```

Hiding a bundle makes its endpoints answer 410 Gone with the problem type `hidden-by-operator` and an explicit interface state, distinct from “unavailable”, because an act of moderation should be visible as one. Hiding a description renders the asset unresolved. Every entry records a reason and a timestamp and is reversible; nothing on chain changes, the asset and its supply remain listed, and any other registry can keep serving the identical self-verifying content. Moderation is exercised at the server over SSH, never through a public HTTP surface, so there is no moderation API to abuse. On the public instance the abuse surface is narrow by construction anyway: image hosting is closed (bundles are text-only there), and the only public write paths are hash-verified resolution and the signed-transaction relay.

6 · BROWSER-SIDE ISSUANCE

Your keys never leave the page

The centerpiece. Hosted issuance has a structural flaw on ZSA: whoever holds the issuance key owns the identity of every asset minted with it, so a service that signs for its users collapses all of them into one issuer and can impersonate any of them later. The only honest design is to move signing into the user's hands, and the browser is where users already are.

Cachet compiles the ZSA protocol stack itself (the QEDIT forks of orchard and librustzcash, with halo2 and the BIP-340 secp256k1 implementation) to WebAssembly, producing a 3.6 MB engine (5.5 MB before binaryen's wasm-opt pass) that runs in a Web Worker. Two builds of it ship: a single-core one that runs anywhere, and a threaded one (rayon over shared-memory wasm threads) the worker prefers on cross-origin-isolated pages; section 7 has the measurements. In the page: BIP-39 seed generation, ZIP-32 issuance and spending key derivation, v6 transaction construction, the Halo2 proof for the mandatory Orchard action, and the BIP-340 issuance signature. The seed lives in page memory only and is forgotten on reload; the page says exactly that next to a checkbox the user must tick. The server receives one thing: the finished, signed transaction bytes.

Notably, no forks were required to reach wasm32: the single-core build routes halo2 through its rayon-free fallback, the threaded build re-enables the same “multicore” path native binaries use (over shared-memory wasm threads), and clang builds the secp256k1 C sources for the wasm target. Both variants are reproducible from one script in the repository.

The relay: a server that cannot cheat

POST /api/v1/relay accepts the signed bytes, checks that the transaction carries an issuance bundle (this is a mint relay, not a general broadcast service), assembles a block from the node's template and submits it, since block production on this testnet is client-driven. The relay can refuse; it cannot alter what was signed, spend anything, reissue anything, or mint under the user's identity, because it never sees a key. For the same reason the endpoint is open on the read-only public instance: read-only means “this instance signs nothing”, and a relayed transaction was signed by the sender's own browser-held key. After relaying, the page teaches the registry its own description through the resolution endpoint of section 3, so the new asset appears with its verified name immediately.

6 · BROWSER-SIDE ISSUANCE, CONTINUED

Measured, end to end

Operation (in-browser wasm)	Measured
Engine load and instantiation	under 100 ms (3.6 MB module)
ZIP 227 descriptor hash	6.4 ms
Issuer identity derivation (ZIP-32 + BIP-340 + Pallas)	7.2 ms
Halo2 proving key construction (single-threaded, per session)	17.3 s
Proof for a 2-action OrchardZSA bundle (single-threaded)	26.1 s
Proving key + proof + signature, threaded build (8 wasm threads)	13.2 s (3.3x)
Proving key alone, threaded (cached per session, warmed during form entry)	7.9 s
Proof + signature with the key already cached	5.6 s (7.7x)
Complete mint from the live page (prove, sign, relay, mine)	30 s threaded · under 2 min single-core

Bench environment: desktop-class consumer CPU, Chromium-based browser, the engine as deployed at cachetzec.com. The end-to-end line is the production mint of “Minted From A Browser”, confirmed at testnet height 520 under issuer key `00de18a2…ae78`, checkable via QEDIT's public RPC. A second live mint from the threaded build (“Eight Threads”, asset `692d5ac5…`, same day) completed in about half a minute, resolution included.

7 · PERFORMANCE

From one minute to seconds: a measured path

Proving cost is the number that decides whether browser issuance feels like a tool or a demo, and the page load around it matters just as much. This section separates what is deployed and measured from what waits on upstream.

Deployed now

Two static passes cut the engine's wire cost by 75 percent: binaryen's wasm-opt shrinks the module from 5.52 MB to 3.58 MB, and zstd/gzip compression at the TLS proxy brings the actual transfer to 1.38 MB, about the weight of an ordinary hero image. The worker is spawned the moment the mint page mounts, so the engine downloads and compiles while the visitor reads the identity step, and it is cached for an hour with a stale-while-revalidate day, so returning visitors skip the download entirely. The result is that by the time a first-time visitor has generated and saved a seed, the engine is ready.

Threads: deployed and measured

Proving is no longer single-threaded. A second engine build rebuilds the Rust standard library with shared-memory atomics and re-enables the multicore halo2 path over a rayon pool of web workers; the console serves the cross-origin isolation headers (COOP/COEP) that shared memory requires, and the mint worker picks the threaded build when the page is isolated, falling back to the single-core build silently anywhere else. Measured on the same consumer CPU with an 8-thread pool: the whole proving pipeline (proving key, proof, signature) drops from 43.4 s to 13.2 s, a 3.3x speedup. The remaining fixed cost was the proving key itself, which the upstream builder rebuilt on every transaction; a thirty-line vendored patch to zcash_primitives (documented in the repository, a candidate for an upstream pull request) memoizes it per process, and the mint studio warms it in the background the moment the user confirms their seed is saved, seconds the user spends typing the asset's name anyway. A mint with the key already cached proves and signs in 5.6 s on the same hardware: 7.7x over the single-core baseline, and the minute-scale mint becomes a ten-second one.

Upstream: the Zakura question

In late August 2026 the Zakura team released Zakura Common, optimized forks of the Zcash cryptography stack reporting 5x faster desktop proving, 14x on mobile and 21x Sinsemilla hashing, consensus-identical and MIT/Apache licensed, already adopted by Zakura's node and by Tachyon. We investigated adopting it for the mint engine the day it was announced. The finding is clean: Zakura Common targets the current generation of the trait ecosystem (ff 0.14, group 0.14), while the entire ZSA stack lives one generation back (ff 0.13), because the QEDIT branches fork from that era. The two cannot be linked into one program. The unlock is a rebase of the ZSA branches onto the new stack, upstream work that will have to happen anyway on the road to mainnet; when it does, Cachet's browser proving inherits the speedup by changing version pins. Stacked on the 7.7x already shipped and measured, one-second browser issuance stops being a slogan and becomes an arithmetic expectation, and we intend to verify it the same way we verified everything else here.

8 · DEPLOYMENT

A public instance built to be worthless to attack

The public instance runs on a small privacy-hosted VPS as four containers behind a TLS proxy: the console, the server, Postgres on the internal network only, and Caddy. Images are built elsewhere and shipped; the box never compiles and never holds source. The server boots with CACHET_READ_ONLY=1 and a throwaway seed generated for this instance alone: the real issuer seed has never touched the machine. Server-side minting, transfers, burns and the wallet endpoint answer 403; the wallet endpoint in particular, because an operator's shielded balances are exactly the kind of information this project exists to keep private.

Rate limiting is per-client (10 requests per second sustained, burst 30, HTTP 429 beyond), keyed on addresses that exist in memory only and are pruned every minute, never logged. Security updates install unattended. The database is dispensable by design: derived state refolds from the chain in about a minute, and the only non-derived table is the issuer's own description journal, which the issuer can reproduce. There are no backups because there is nothing to back up.

What a full compromise yields

An attacker who fully owns the public instance can…	…and cannot
Deface the website and serve wrong pages (until redeployed)	Forge metadata past verifying clients: bundles re-hash in the browser
Refuse to serve: bundles, names, the relay (availability loss)	Steal funds or assets: the throwaway seed holds nothing and signs nothing
Read the same public chain data anyone can read	Learn holders, balances or transfers: the instance never has them
Spend the instance's worthless testnet seed	Mint as any real issuer: asset ids derive from keys it does not have

9 · LIMITATIONS

What this is not, on purpose or not yet

By design. Cachet is not a marketplace and will not become one; it takes no custody at any phase, runs no accounts, no allowlists and no ownership database, and hosts no third-party images on the public instance. It does not attest real-world issuer identity. It makes no mainnet claims: everything here runs on a test network whose assets carry no value, and anyone who tells you otherwise about any ZSA project today is lying to you.

Not yet. Threaded proving needs a cross-origin-isolated browser context; elsewhere the mint studio falls back to the single-core build and an honest half-minute wait. Browser mints carry text-only metadata for now; sealed images from the browser need an abuse story we are not done designing. The mint relay serves issuance transactions only; browser-side transfers and burns are a natural next step on the same architecture. ZMD-1 recognition covers descriptors, not yet full-form manifest verification. Registry snapshots (signed exports a mirror can serve offline) are specified but unbuilt. The stack rides alpha protocol forks pinned by exact revision, and tracks them deliberately slowly.

Roadmap, gated on events rather than dates

Gate	What ships when it opens
Application work (no external gate)	browser transfers and burns over the relay; ZMD-1 full-form verification; signed registry snapshots
ZSA branches rebase onto the current crypto stack	Zakura Common speedups inherited by pin change; the seconds-scale browser mint measurement
ZSA activates on Zcash mainnet	Cachet retargets by configuration: same registry, same verification, same browser signing, real assets

REFERENCES

Citations and pointers

ZIP 226	Transfer and Burn of Zcash Shielded Assets · zips.z.cash/zip-0226
ZIP 227	Issuance of Zcash Shielded Assets · zips.z.cash/zip-0227
ZIP 244	Transaction Identifier Non-Malleability · zips.z.cash/zip-0244
OrchardZSA audit	Least Authority, security audit of the OrchardZSA implementation, final report January 2025
QEDIT reference stack	github.com/QED-it: orchard, librustzcash, zebra (ZSA branches); the public testnet node at dev.zebra.zsa-test.net
ZSA testnet announcement	Zcash community forum, August 2026
ZMD-1	ZecBit metadata convention, as published in the ZecBit technical whitepaper v1.0, August 2026; Cachet implements descriptor recognition
Zakura Common	zakura-core/common, high-performance Zcash cryptography, announced 29 August 2026
Halo 2	zcash/halo2; proving system used by Orchard, no trusted setup
Cachet	cachetzec.com · api.cachetzec.com/api/docs · @Cachet_zec · source to be published under MIT

This document describes software, not an investment. Nothing here is financial advice, no token exists or is planned, and testnet assets carry no monetary value. The text will be revised as measurements change; the version number on the cover is the authority on which claims were current when.