STYX PROTOCOL · WHITE PAPER

Styx: private 1 SOL payments on Solana, checked on chain by a hash-based proof

16,500 words · 72 min read · In English

CONTENTS

1. Summary for a Solana user

How to read the rest of this paper

2. Threat model: what is hidden, and from whom

2.1 Chain observer (anyone reading Solana)

2.2 RPC provider (Helius today)

2.3 Our server and operator (till, float, issuer, key-value store; sections 3.2 and 3.3)

2.4 Device thief

2.5 Quantum adversary (a large quantum computer, if one is built)

3. How it works

3.1 Notes and the 1 SOL pool

3.2 Contribute one, collect an older one

3.3 The float and the till

3.4 Proofs made in your browser

3.5 The verifier on chain

3.6 The circuits actually live

4. Cryptography and security levels

4.1 Parameters

4.2 The regimes

4.3 v1 figures, as deployed

4.4 What one-element fingerprints and base-field challenges cost v1

5. Known limits of v1

6. The internal audit

6.1 What it is, and what it is not

6.2 Method

6.3 What it covered and did not

6.4 Results

6.5 What comes next

7. Performance

7.1 Method

7.2 Results

7.3 Comparison with other systems, on two axes

8. v2: the decisions and what each is designed to fix

9. Reproducibility annex

9.1 Rebuild the prover byte for byte

9.2 Re-run the calculator

9.3 Re-run the tests

9.4 Re-run the benchmark

9.5 Access to the code and licence terms

White paper, first edition, 2026-09-23. Volta Team, https://styx.cash.

Status in one line. Styx runs on Solana **devnet only**. The version described here (v1) has a known critical flaw that lets one deposit be withdrawn twice, it has had an internal review by AI agents and no external human audit, and it will never go to mainnet. The next version (v2) is decided and not deployed: two of its building blocks (the new hash, and the new field and transcript for the verifier's random challenges) are written and tested, but are not yet in the published repository, and nothing of v2 runs on chain. This paper says what v1 does today, what it gets wrong, and what v2 is designed to change. [K1, K2, K3, K4, K129]

Section 1 is written to be read on its own. The conventions for the rest of the paper follow it.

---

1. Summary for a Solana user

The problem. On Solana, anyone can see who paid whom, how much and when, forever. If you pay a merchant from your main wallet, the merchant and everyone else can read your balance and your history.

What Styx does. Styx lets you turn 1 SOL into a private *note* (a secret receipt kept in your browser that says "this is worth 1 SOL in the pool"), then later spend that note: withdraw it to a fresh address, pay a subscription with it, or hand it back in exchange for another note (that exchange is switched off in the web app since 2026-09-23 because of audit finding F70, section 3.2). When you spend, your browser computes a mathematical proof that you hold a valid note in the pool, and publishes a one-time tag meant to stop the same note being spent twice. A program on Solana checks that proof itself before any SOL moves. In v1 that protection has a hole: one deposit can be withdrawn twice (audit finding F03, below). [K5, K6, K7, K43, K18, K131]

What it hides, and from whom. Every note in the pool is worth the same 1 SOL, so amounts do not tell users apart. [K8] A spend on the web app's main spend path (circuit C7; a circuit is the fixed set of rules one kind of proof checks) publishes no *commitment* (the note's fingerprint, below), so nothing on chain names the deposit it came from. Three limits apply: we have not yet established that the proof itself reveals nothing about the note (audit finding F69, section 4.4), the pool snapshot (the tree root) a spend names can hint at when its note was deposited (section 2.1), and the spend names which part of the pool's tree, of at most 2,048 notes, holds its note, so the note hides among that part's notes only, and is alone in it when that part has just started to fill (section 2.1). [K9, K71, K122, K148] When you press Shield (the button that turns SOL into a note), by default you do not deposit from your wallet: you pay the operator (whoever runs the Styx web app), and it hands you a note it deposited earlier (the *default Shield path*). Someone reading the chain then sees your payment and your later spend attached to two different notes. The operator is not fooled: it knows which note it sold you (next paragraph). [K10, K14] Until 2026-09-22, if the note-issuing server did not answer, the Shield button could deposit from your wallet without asking, which put your wallet on the deposit for everyone to see. The fix went live in the web app on 2026-09-22:

when that server does not answer, the app now charges nothing and offers only a deposit from your wallet under a clear label (or you press Shield again to ask again). Until 2026-09-23 that labelled deposit button lost about 1.013 SOL per click (audit finding F56, section 3.2); a web fix went live on 2026-09-23. A check of the live site that both fixes behave as tested is still to be done. [K120, K47, K130]

What it does not hide. Deposits are public. [K11] Your RPC provider (the server your wallet and browser use to talk to Solana; Helius today) sees your IP address and your requests. [K12] If you spend right after buying, the timing can link the two, and with roughly three to four note purchases a day on the 1 SOL pool, the crowd you hide in is small. [K13] The operator running the Styx web app can compute, and therefore spend, the notes it issues to you, and it can recognise their spend and link it to the wallet that paid: with the default Shield path you trust the operator with your note until you spend it, and with your privacy from the operator after that. [K14, K31] Anyone who obtains your wallet's private key can re-derive every note your browser made from it (the notes you deposited yourself), past ones included [K15, K34]; a note the operator issued you comes from the operator's own seed, and its only copy is in your browser (section 2.4) [K14, K35].

Why it is different. The proof is built from hash functions only (SHA-256, and Poseidon, a hash built for proofs). It uses no elliptic curves (the mathematics behind today's wallet signatures) and no secret setup ceremony (a one-time event whose leaked secrets would let someone forge proofs). A Solana program we wrote for that purpose checks it in full; no off-chain committee or hardware enclave (a sealed chip you must trust) does it. [K16, K17] A large quantum computer, if one is built, could forge today's Solana wallet signatures; it is not known to break hash-based proofs like this one. [K38, K39] But v1's proofs have weaknesses that need no quantum computer (sections 4 and 5). The mathematics we rely on (a published theorem: the Johnson regime, as proven in the paper known as BCIKS20, section 4) only proves that forging a v1 spend proof (C7) takes at least some 50,000 attempts (about 2^15.6), or about 220 (about 2^7.8) for a quantum attacker. That does not mean forging costs that little; it means the proof of safety stops there (the cheapest known attacks are described below). v2 is designed to raise these floors (section 8), and even then the notes your browser derives itself, and the keys of your stealth addresses, come from your wallet signature, which a quantum computer would break (section 2.5). [K66, K108, K15, K38]

Where it stands: a double spend. An internal audit run by AI agents confirmed 79 findings on v1, one of them critical. The fingerprint that identifies each note (its *commitment*) is a single 64-bit number, so it can take only about 18 billion billion values: few enough that a desktop computer can find two notes with the same fingerprint in under half an hour, and two such notes let one deposit be withdrawn twice (finding F03). An audit agent found such a pair in under 28 minutes on 16 threads of one desktop and had both withdrawal proofs accepted by the code of the Solana program that checks proofs (the verifier), run on its own machine; no transaction was sent. [K18, K19]

A forgery. A second finding, rated high (F52), lets an attacker forge a proof of any statement by trial and error, re-rolling the verifier's random challenges (the random spot checks the

verifier makes on each proof) until one lands where the forger wants: about 2^51 attempts, which is about 186 to 236 days on 24 threads of the audit's desktop, or about one day on some 187 to 237 such desktops, since the search splits across machines. Applied to a spend, a false statement is a withdrawal with no note behind it, so a completed forgery could withdraw SOL that was never deposited; that consequence is our reading, since no complete forgery was run. [K20] Both findings come from one root cause: v1 packs its fingerprints and its random challenges into single 64-bit numbers, where the design needs far larger ones. [K21]

v2, and the next step. That is why v1 stays on devnet, and why v2 exists. v2 is designed to remove that root cause: to use four 64-bit numbers instead of one for each fingerprint, and to draw its random challenges from a much larger set of values. On estimated parameters, v2 is designed so that the same theorem (Johnson regime, BCIKS20) would prove that forging takes at least about 2^105 attempts (105.42, a design figure on estimated parameters, not a property of a built system), against about 2^15.6 for C7 today; for a quantum attacker, the same design figure is 52.71 (about 2^52; section 8). Each +1 in the exponent doubles the work: 2^105 is about 2^71 times the 2^33.59 hash evaluations of the F03 collision search (under 28 minutes), above. That design figure has still to be re-run with the inputs of the current proof-format design and checked on the real circuit; no v2 figure holds until it is (section 8). The new hash and the new challenge field are written and tested outside the published repository; they are not deployed (section 8). [K21, K22, K23, K108, K109, K149, K19, K129] The next step is an external review by independent human cryptographers and Solana auditors. [K24]

How fast, and what to do. On devnet, a withdrawal takes a median of 32.3 s on our test machine (30 runs), from the start of the withdrawal until Solana confirms it, running the app's own code in a test harness on our machine (not a browser), with the one-time key funded directly by the measurement wallet rather than the operator's float, and the pool's history already cached, as for a returning user (a first visit, with nothing cached, was not measured for this edition; section 7). Most of that time is waiting, not work: summed over those 30 runs, 63.2% of it was spent waiting on the rate limit of the free plan of our RPC provider (Helius), a limit of the RPC plan the app uses today, which users of the live app also wait on, not work done by Styx's code; no run was made on a plan without that limit (section 7.2). Devnet SOL has no value: use Styx to try private payments, not to hold money, until v2 and an external review. Reviewers can start from section 9. [K1, K88, K89, K143, K146]

How to read the rest of this paper

A tag such as [K12] points to row K12 of `docs/CLAIMS.md`, which names the evidence for that fact: a test, a log, an output of the project's security calculator (`tools/security-levels`, which computes the security figures from the verifier's source, section 4.1) or an audit finding.

Security figures are base-2 logarithms of the work a cheating party must do, and each one is written next to the regime it holds in; a figure without its regime is not a figure from this paper. For scale: a figure of 40 means about 2^40 (about a trillion) attempts, and each +1 doubles the work. The F03 collision search, at the 32.00 level (generic classical collision bound, birthday; log2 of hash evaluations), took under 28 minutes (1,666 seconds) on 16 threads of one desktop. [K19]

Audit findings are written F01 to F79 (the internal audit's numbering). Rows of the project's leak ledger, `docs/LEAK-LEDGER.md`, are written "ledger B6", "ledger E3" and so on. The ledger marks a fix "closed in the working tree" (changed in our local copy of the code only: not yet committed, that is, not yet recorded in the project's shared history, and not deployed) when it is written and pinned by a passing test; such a fix counts for users only after a deploy and a probe of production. The ledger's status column was last updated on 2026-09-20, before the web deploy of 2026-09-22, so for the web fixes this paper cites (ledger B11, B12, C7, D3, E9) we state the production state read from the repository and from the hosting platform instead: live since 2026-09-22, not yet confirmed by a post-deploy probe. Anything we have not measured is marked as an estimate or as not measured.

"The operator" is whoever runs the Styx web app and its server: the till, the float, the note issuer and the key-value store (section 3).

Fix status. The block in section 6.4 is the reference for the fix status of the audit's findings. The few other places that state a fix status (listed in `docs/CLAIMS.md`, section "Where fix status and benchmark figures are stated") are updated together with it.

---

2. Threat model: what is hidden, and from whom

We name five observers. The table gives each in one line; the subsections below say what each learns from v1 as it runs today, with the evidence.

OBSERVER	IN ONE LINE
Chain observer (anyone reading Solana)	sees every deposit and every spend; on the C7 path the instruction carries no commitment, but it names the part of the tree (at most 2,048 notes) the note sits in; F69 not settled and the root can date the deposit (2.1)
RPC provider (Helius today)	sees your IP address and your requests (2.2)
Our server and operator	knows every note it issues, so it can spend it and recognise its spend; sees the requests its float and relay serve, spends of notes you deposited yourself included (2.3)
Device thief	gets what the browser stores, and with your wallet key every note you deposited yourself, past ones included (2.4)

OBSERVER	IN ONE LINE
Quantum adversary	breaks wallet signatures; the proof rests on no curve, but v1's floors are low (2.5)

2.1 Chain observer (anyone reading Solana)

- **Sees.** Every deposit in full: the depositor, the fingerprint (commitment), its position in the pool's Merkle tree (a tree of hashes whose single top value, the root, summarises every note in it), and the new tree root [K11]. On a spend: the nullifier (a one-time tag meant to stop double spending), the payee, the one-time fee payer, the tree root the proof names, and the envelope (proof size, compute used), which reveals the kind of operation [K25, K26]. A C7 spend also publishes the top of the note's path in the tree: the pool's tree is 15 levels deep and the proof covers the lowest 11, so the spend names which of 16 subtrees of at most 2,048 notes (2^11) holds the note, and the note hides among that subtree's notes, not the whole pool (ledger B4, open). When a new subtree starts to fill (at the 2,049th note, the 4,097th and so on), its first note is alone in it until the next deposit. While the pool holds 2,048 notes or fewer, every note is in the first subtree [K148]. The pool's state: its note count, a histogram of deposits over 32 epochs (fixed windows of 7,200 slots, about 48 minutes each at 400 ms a slot) and its ring of recent roots (the last few snapshots of the pool, any of which a proof may name) [K27].
- **Also sees, through links still open.** The root a spend names can date the note's deposit (ledger B11). The web fix (a spend proves against the root it has just read) went live on 2026-09-22 and is not yet confirmed by a post-deploy probe; the channel stays open for every client until the pool program stops accepting older roots from its ring (today it accepts any of them) [K122]. Until 2026-09-22, in the web app, a blinded note (one whose fingerprint mixes in a secret number, its blinding value, derived from your wallet's pool seed, or from the operator's seed for a note it issued, so that recomputing the fingerprint from public data means guessing that number; see "The limit" below) [K42, K28, K15] whose C7 pre-flight check (a local test run of the proof before sending) failed could silently fall back to the older two-proof spend path (C1+C3, section 3.6) and publish its commitment; 30 of 64 landed spends used that older path, and how many came through the fallback is not measured. The web fix went live on 2026-09-22, not yet confirmed by a post-deploy probe; the extension and the mobile app keep their own routing (ledger B12) [K123].
- **Does not see.** On the C7 path, a commitment: the instruction carries none, so nothing on chain names the deposit a spend consumes [K9]. Three caveats: that the proof itself reveals nothing about the note is not yet established (audit F69, section 4.4) [K71], the root the spend names can date the deposit (B11, above), and the spend names the subtree the note sits in, so the deposit is one of that subtree's notes, and the only one when the subtree holds a single note (B4, above) [K148].
- **The limit.** From the published values alone, and assuming the proof reveals nothing about the note (not yet established, F69), matching a spend to its deposit means

guessing the note's blinding value, drawn from 2^63 possibilities, for an expected 62.00 (classical search, log2 of hash pairs) or 31.50 (Grover, the generic quantum speed-up for search) [K28]. By our arithmetic, that is about 2^28 times the work of the F03 collision search, or about 18,800 years at the speed of that search on 16 threads of one desktop. That figure is for one spend and one desktop: the search splits across machines, and the speed of Poseidon on graphics cards was not measured, so it gives a scale, not a safety margin [K124]. Notes deposited before the blinding value was introduced, and the older C1+C3 path, publish their commitment and are linked by reading it [K29]. Proofs made before masking was added stay on chain and can give up their secrets (section 4.4) [K147].

2.2 RPC provider (Helius today)

- **Sees.** Your IP address and every request your browser makes, for both the purchase and the spend, from the same IP [K12]. At spend time, a fresh one-time key pays the spend's fee, and the operator's float (a funding wallet) or its relay (an operator route that forwards SOL you send it to a fresh address of yours, so the two addresses never appear in one transaction; amount and timing can still link them, and the operator sees both, ledger E3) funds that key (section 3.3) [K121, K128, K41, K32]. The client's anti-link check (a test that your wallet and that funder have never appeared in the same transaction) then asks the RPC about both addresses together, so the RPC sees them side by side (ledger D2, open) [K121].
- **Does not see.** Anything your requests do not carry. In the web app live since 2026-09-22, the client asks about the whole pool rather than about one note when it exports a note, withdraws a received note or prepares a subscription, so those questions do not point at your note; not yet confirmed by a post-deploy probe (ledger D3). The extension still asks about the note itself (ledger D3, open for the extension) [K30].

2.3 Our server and operator (till, float, issuer, key-value store; sections 3.2 and 3.3)

- **Sees.** The wallet that paid, the claim code it received (the code redeemed for a note, section 3.2) and the note address it was issued to, kept without expiry [K31]. The issuer's seed derives every note it issues, so the operator can recompute and spend any issued note until you spend it. It can also compute that note's nullifier, so it recognises the spend when it lands and links it to the wallet that paid: against the operator, an issued note hides in a crowd of one [K14, K31]. One operator runs issuance, the funding of one-time keys and the spend relay, which gives it an off-chain map of all three [K32]. The programs' upgrade authority (the key that can replace the programs' code, so whoever holds it can change the rules of the pool) is an operator key that also serves as treasury and relayer key [K33].
- **Does not see, and the limit.** For notes you deposit yourself, their *opening* (the secrets that let someone spend a note) is yours alone [K34]. The proof and the on-chain C7 spend of such a note tell the operator nothing beyond what a chain observer sees [K9,

K34]. But whenever that spend goes through the operator's float (which funds the one-time key) or its relay, the operator also sees those requests, what they ask for and when (ledger E7; the RPC side of the same spend-time funder check is ledger D2, section 2.2) [K32, K121].

2.4 Device thief

- **Sees.** Whatever the browser stores. In the default Shield path, the only copy of an issued note has been a browser storage entry [K35]. Some records are stored in clear (a buyer key, relay receipts), and the encrypted records are not protected against tampering [K36]. With your **wallet key**, the thief re-derives every note your browser derived from it (the notes you deposited yourself), past ones included: the web pool seed is derived from one wallet signature [K15]. A note the operator issued you is not derived from your wallet; its only copy is the browser storage entry above, so the thief gets it only while that copy is still stored [K14, K35]. A passphrase-salted derivation exists in the code, and no screen of the web app sets it today [K37].
- **Does not see.** Nothing more than the device holds; there is no server-side account to break into.

2.5 Quantum adversary (a large quantum computer, if one is built)

- **Breaks.** Wallet signatures on Solana are Ed25519, which Shor's algorithm (the quantum method that breaks elliptic-curve keys) breaks. That gives a quantum attacker spend authority over your wallet and every key the web client derives from a wallet signature: the pool seed, and the keys of stealth addresses (one-time receiving addresses). Stealth addresses combine X25519 with ML-KEM, a key exchange designed to resist quantum computers, but their keys come from the wallet signature, so they fall with it [K38]. The v1 note fingerprint gives a generic quantum collision bound of 21.33 (Brassard-Høyer-Tapp, the best generic quantum method for finding collisions), against 32.00 classically, the level the audit beat in under 28 minutes [K21, K19].
- **Does not break, and the limit.** The proof system itself rests on no curve, so Shor's algorithm does not apply to it. Its quantum line is half the classical figure in each regime: a floor given by the security proof for how the random challenges are derived from hashes (the Fiat-Shamir step), not a known attack [K39]. For v1 those floors are low: 7.82 to 8.14 in the Johnson/BCIKS20 regime (section 4.3). A floor that low means the security proof only rules out a quantum forger who makes fewer than about 220 to 280 attempts (2^7.82 to 2^8.14); it is not an attack [K66].

Accepted residuals. The founder has accepted four residuals as design choices, and the audit excluded them from its findings [K40]:

Helius sees IPs. Like any Solana wallet that uses an RPC provider. The client avoids per-note questions where it can (see 2.2).

The 1 SOL denomination. Everyone moves the same amount, so the amount cannot tell users apart. This is uniformity, not a leak.

Deposits are public. The pool is fed by public deposits; privacy comes from the spend not naming which one.

Purchase-to-spend timing. If you buy a note and spend it at once, the two events sit next to each other in time. With about three to four note purchases a day, the busier the pool and the longer you wait, the harder that link is to draw.

Links still open in the ledger. Among the further links recorded as open: the till's settlement to the float is a public transfer (ledger E2), purchase and spend are not decorrelated on the relay path (ledger E3), an abandoned subscription attempt and a later withdrawal can share a one-time key (ledger E4), and the notes the operator stocks can be told apart by their funder, their cadence and the IP ranges of the restock job, so the pool's note count overstates the crowd (ledger E5). [K41]

---

3. How it works

3.1 Notes and the 1 SOL pool

A note is a small set of secrets. Its *commitment* (the fingerprint of section 1; for a note spent on circuit C7, a Poseidon hash of the note's secrets, a blinding value and the token) goes into a Merkle tree held by the pool program `zk_shielded` (`GbVM5yve…`). [K42] For a note you deposit yourself, only you know the secrets behind its commitment; the operator also knows the secrets of every note it issues. [K34, K14] Spending a note publishes its *nullifier*, a value derived from those secrets: the program records every nullifier and refuses one it has seen, which is meant to make each note spendable once, though in v1 one deposit can still be withdrawn twice, because two different notes can share one commitment (F03, section 5). [K43, K18] The web app uses the 1 SOL pool; a deposit costs 1 SOL plus a 0.3% fee and a withdrawal pays 0.995 SOL (a 0.5% fee). [K44, K8]

A note on version numbers: v1 and v2 name the protocol; v3, v4 and v5 are version numbers of the pool program's instructions and accounts. Solana's own "transaction v1" format is called "larger transactions" here.

3.2 Contribute one, collect an older one

The web app's **Shield** button (the button that turns SOL into a private note) does not, by default, deposit from your wallet. Instead: [K10, K45]

You pay 1.003 SOL to the **till**, the operator's collection wallet.

You get a claim code.

You redeem it for an **older note** from the operator's **inventory**: a note the operator deposited into the pool earlier, from its own key.

Your payment funds a fresh note for that inventory, deposited at once by a one-time key in the *contribute* path, or at the next scheduled restock. You never learn that new note's opening.

Because the note you walk away with was deposited earlier by the operator, a chain observer sees your payment and your later spend at the ends of two different notes. The links that remain are timing (residual 4) and the operator's own knowledge (section 2.3). [K13, K14] Until 2026-09-22, a failure of the note-issuing server could turn step 1 into a deposit from your wallet, without asking. The web app live since 2026-09-22 charges nothing in that case: it shows a notice whose only button is a deposit from your wallet, labelled as such (pressing Shield again asks the server again); when the server answers that no older note can be handed over yet, the notice also offers to continue on the default path. A post-deploy probe has not yet confirmed it (ledger E9). The deposit it offers is the "Deposit my own note (linked to my wallet)" button, which the audit found loses about 1.013 SOL per click (F56, below); its web fix has been live since 2026-09-23, not yet confirmed by a post-deploy probe. [K120, K47, K130]

The **note-in exchange** is the same idea for someone who already holds a note: you withdraw it to the till on circuit C7 and receive an older issued note in return. Since 2026-09-23 it is switched off in the web app because of audit finding F70 (below). [K46, K131]

Problems the audit found in these paths, and where they stand. Three high audit findings touch them. Web changes for all three were committed and went live on 2026-09-23; none has been confirmed by a post-deploy probe, and none has been re-examined by the audit's own three-skeptic process (section 6.4). [K47, K72, K130, K131]

- The "Deposit my own note (linked to my wallet)" button paid the till and then refused to relay, losing about 1.013 SOL per click (F56). The button now passes the wallet's message signer to the relay, which is what the relay needed to accept the payer; a test that failed before the change passes after it.
- In the note-in exchange, whoever pays the fee of the withdrawal to the till gets the claim code, and someone who copies the proof can become that payer (F70). The exchange is now switched off. The web client refuses it before anything is spent unless a build flag turns it back on, and the claim route sells no note against a withdrawal to the till unless that withdrawal landed before a cutoff the operator sets; neither the flag nor the cutoff is set in production (the project's environment variables, read on 2026-09-23). The refusal happens on the client before anything is spent, but the claim route refuses only after the withdrawal to the till has landed: a caller that skips the web client's refusal (a page loaded before the deploy of 2026-09-23, or any other caller of the route) sends its note's 0.995 SOL to the till and gets no note, and the route's answer offers settlement through support only for withdrawals that landed before the switch-off. Our benchmark trial did exactly that three times (section 7.2). The flaw itself is not fixed: it needs a different claim identifier, or a C7 proof that names the submitter (a redeploy).

- In the contribute path, a stranger whose request names the same position in the pool's tree could take the claim of a successful contribution (F72). A contributor's hold on that position now lasts while its deposit can still land, and at most 3 hours by default; past that ceiling the takeover the audit described is still possible, so this is a mitigation, not a fix.

The note-in exchange described above is therefore a design, not a path a user of the web app can take today. [K46, K131]

3.3 The float and the till

A spend needs a fee payer, and a one-time key cannot pay fees from nothing. The operator's **float** (a funding wallet) sends each one-time key what it needs, so that your wallet does not appear as the payer. The till's takings go back to the float in batches, after a quiet period and a random hold, and from there to the wallet that restocks the inventory. [K48] The settlement from till to float is a public transfer (ledger E2). [K41] In the web app live since 2026-09-22, a spend whose funder refuses stops before anything is sent, instead of silently falling back to your wallet; a post-deploy probe has not yet confirmed it, and the extension and the mobile app keep their own funding (ledger C7). [K49]

3.4 Proofs made in your browser

Your browser runs the prover, a WebAssembly file of 240,172 bytes (SHA-256 `241caaab…`), in a background worker. Since 2026-09-23 it exports proving for the five circuits the apps use (C0, C1, C3, C6 and C7) and no longer for C2, C4 and C5 (section 3.6). [K50] It builds a STARK proof (a proof system that uses only hash functions): the statement ("I know a note in this tree whose nullifier is this, and I want the money sent here") is written as a table of numbers that must satisfy fixed rules, the table is padded with random values (a mask) drawn from the operating system's random generator, and the proof shows that the rules hold while opening only a few randomly chosen entries of the masked table. [K51, K52] A circuit-7 proof is 79,405 bytes. It is uploaded to a proof buffer account in 3,840-byte pieces (21 for a C7 proof), each carried by one of Solana's larger 4,096-byte transactions (SIMD-0385, active on devnet), when the uploader finds that feature active. Where it is off, the uploader falls back to 1,000-byte pieces in ordinary transactions. [K53]

3.5 The verifier on chain

The verifier program `p01_stark_verifier` (`DGY37k3J…`, 801,457 bytes, redeployed on devnet on 2026-09-12) was written for Solana, with no proving library at run time. It recomputes every random challenge from a SHA-256 transcript of the proof, checks the proof's openings against its Merkle roots, checks the rules at a random point (DEEP-ALI), and checks that the committed data is close to a low-degree polynomial (FRI). [K17, K54] A circuit-7 proof from the shipped prover was accepted on devnet on 2026-09-23 in slot

502,692,190, with both verification phases in one transaction: 889,882 compute units for the first phase and 193,269 for the second, 1,083,301 for the transaction, under Solana's 1,400,000 limit for one transaction. The verifier was not redeployed for it. [K55] The pool instruction then consumes the verified buffer and moves the SOL. [K56]

3.6 The circuits actually live

The deployed verifier accepts eight circuit identifiers; since 2026-09-23 the shipped prover exports five, the five wired into the apps (C0, C1, C3, C6, C7). The web app runs on C6 (deposits) and C7 (spends). Its C1+C3 path for older notes is switched off since 2026-09-23 (audit finding F05, section 5) unless a build flag turns it back on, and production does not set that flag (read on 2026-09-23). The mobile app's proofs are rejected today (its published build carries an older prover, section 5), and no user has made a C0 proof since the verifier's redeploy of 2026-09-12; the only accepted masked C0 proofs (masked, meaning padded with random values meant to make them reveal less about the note; what that achieves is in section 4.4) are benchmark proofs, made with the measurement key: the harness run of 2026-09-12 (slot 497,239,840) and the 30 proofs of the proof-pipeline run of 2026-09-23 (`run-20260923T032635Z`, section 7.2). [K57, K58, K73, K131, K144] The last column of the table below leaves out the benchmark and test proofs of 2026-09-23 (sections 3.5 and 7.2): on that day the measurement key also had C1, C3, C6 and C7 proofs accepted by the verifier, and made C6 deposits and C7 withdrawals in the pool. [K55, K58, K144]

CIRCUIT	WHAT IT PROVES	USED BY	LAST USED ON DEVNET, BENCHMARK AND TEST PROOFS OF 2026-09-23 EXCLUDED
C7 spend	a note in the tree is spent to a named payee	withdrawal (unshield v4) and subscription (subscribe v4); the web app's main spend path	2026-09-22 [K58]
C6 merkle_update	a new commitment is inserted correctly	every deposit (shield v3)	2026-09-22 [K58]

CIRCUIT	WHAT IT PROVES	USED BY	LAST USED ON DEVNET, BENCHMARK AND TEST PROOFS OF 2026-09-23 EXCLUDED
C1 + C3 (pool commitment + Merkle path, always together)	the holder knows a note, and the note is in the tree: the older spend path, in two proofs	web: fallback for notes made before the blinding value was introduced, switched off since 2026-09-23 (F05); the mobile app (whose published build's proofs the chain rejects today) and the extension's transfer	2026-09-17 [K58, K131]
C0 subscriber_ownership	the owner of a private subscription pauses or resumes it	extension and mobile, never the web	no C0 proof from any user since the redeploy of 2026-09-12 (none in the verifier's last 1,000 transactions, to 2026-09-22); the only accepted masked C0 proofs are benchmark proofs: the harness run of 2026-09-12 (slot 497,239,840) and the 30 of `run-20260923T032635Z` (section 7.2); last user pause on 2026-06-17, last resume on 2026-05-11 [K58, K144]

C2 (balance proof), C4 (confidential balance) and C5 (transfer) are still accepted by the deployed verifier, and their rules stay in the Rust prover crate for that reason, but no deployed instruction consumes their proofs. On 2026-09-23 they left the product: the shipped prover no longer exports them, and the legacy mobile screens that proved C4 and C5 were deleted from the repository. [K59, K132] The older Circom/Groth16 circuits (Groth16 is a curve-based proof system that needs a setup ceremony) are verified by no program. Their circuit, key and setup files were deleted from the repository on 2026-09-23 (none is tracked at `c0865f89`), and so was the old extension archive the web app served; the three tests that ran Groth16 proofs with `snarkjs` and the root `snarkjs`, `circomlib` and `circomlibjs` dependencies leave in the commit that publishes this paper. They remain in the repository's history and in copies downloaded earlier. [K60, K132]

What was removed on 2026-09-23. Code that no shipping flow used was deleted from the repository: the Arcium MPC program and SDK (`p01_arcium`), the zkSPL program and SDK, the Groth16 circuits and provers of the extension and the mobile app, the `stream`, `subscription`, `whitelist` and `p01_liquidity` programs, and eight unused packages. On devnet, the program-data accounts of these deployments no longer exist (the programs were closed), so they cannot run: `p01_arcium` at `FH1JiQRU…` and an older `p01_arcium` at `9kMjmVMY…`, older deployments of `stream` (`2yH26XmX…`) and `subscription` (`5kDjD9LS…`), and the

old Groth16 zkSPL program (`EqppogLB…`). `p01_liquidity` (`6PfFkvjX…`) is still deployed on devnet; its pool was set inactive on 2026-09-22 (audit finding F27, section 5). The repository now declares four programs: the pool `zk_shielded` and the verifier `p01_stark_verifier`, which carry every flow of this paper, the merchant registry `p01_registry`, and the relay program `p01_relayer`, which is deployed but operated by no hosted node. [K133, K134]

---

4. Cryptography and security levels

This section is for reviewers; the plain-words version is in section 1.

4.1 Parameters

Every figure in this section comes from `docs/SECURITY-LEVELS.md`, which the calculator in `tools/security-levels` generates from the verifier's own source files: it compiles `compact_proof.rs` and `goldilocks.rs` rather than copying their numbers, and a test fails if the document drifts. [K61]

- **Field.** Goldilocks, p = 2^64 − 2^32 + 1; every random challenge in v1 is one element of this field (no extension). [K62]
- **Proof.** DEEP-ALI over a batched FRI with folding by two; blowup 16, with rate 1/16 enforced by the verifier on every circuit; 22 queries (27 on C1 and C2); a proof-of-work of 22 leading zeros before the queries ("grinding"). [K63]
- **Hashes.** SHA-256 for the proof's Merkle trees and the Fiat-Shamir transcript. Poseidon over Goldilocks (width 3, S-box x^7, 30 full rounds) for note commitments, nullifiers and the pool's Merkle tree; its output is **one** field element. [K64]
- **Stealth addresses.** A hybrid X25519 + ML-KEM-768 key exchange, whose keys are re-derived from the Ed25519 wallet key. [K38]

4.2 The regimes

A security figure depends on which mathematical result backs it. The calculator reports five, and we quote them together. In plain words: the theorem columns rest on published proofs; the conjectured column rests on an unproven assumption that has been partly refuted, so we never quote it alone. [K65]

> **Unique decoding (theorem).** Proven for proximity just inside the unique-decoding radius. On v1 it is not the lowest figure: it sits above the Johnson/BCIKS20 figure on every circuit (4.3). [K67]
>
> **Johnson, BCIKS20 (theorem).** BCIKS20 is IACR eprint 2020/654. The calculator's headline column and the one v2 is designed against.

Johnson, BCHKS25 (theorem, 2025). BCHKS25 is IACR eprint 2025/2055. A newer and tighter result, shown for information.

Conjectured (ethSTARK / BCIKS20 Conjecture 8.4). Assumes proximity gaps up to list-decoding capacity. Counterexamples to strong forms of that conjecture were published in 2025 (Diamond and Gruen, IACR eprint 2025/2010; Crites and Stewart, IACR eprint 2025/2046; BCHKS25), so this column is never quoted alone.

Quantum. Half the classical figure of its regime (Chiesa, Manohar and Spooner, TCC 2019), capped at 85.33 by SHA-256. A floor the proof gives, not a known attack.

4.3 v1 figures, as deployed

For the circuits that serve users, as shipped (the verifier's challenge sampler included, see 4.4); union bound over every error term, truncated to two decimals. [K66]

CIRCUIT	UNIQUE DECODING (THEOREM)	JOHNSON, BCIKS20 (THEOREM)	JOHNSON, BCHKS25 (THEOREM, 2025)	CONJECTURED	QUANTUM: UD / JOHNSON BCIKS20 / CONJ.
C0	42.01	16.28	31.84	46.91	21.00 / 8.14 / 23.45
C1	45.85	16.28	31.81	46.75	22.92 / 8.14 / 23.37
C3	42.00	15.97	31.52	46.75	21.00 / 7.98 / 23.37
C6	41.99	15.64	31.20	46.75	20.99 / 7.82 / 23.37
C7	41.99	15.64	31.21	46.91	20.99 / 7.82 / 23.45

How to read it. Each column is a separate bound. Each figure is what a theorem (or, in the conjectured column, a conjecture) guarantees, not the price of a known attack: a low figure means the proof of safety is weak, not that forging costs that little. The cheapest known way to make the verifier accept a proof of a false statement is F52, at about 2^51 draws (4.4). Cheaper attacks on v1 exist that use proofs of statements the circuit's rules do accept: F03, two notes with one commitment (4.4); F01, a deposit proof whose rules leave the insertion position free; and F05, re-submitting copied C1+C3 proof bytes (section 5). [K66, K126]

The largest v1 figure in a theorem regime is 45.85 (C1, unique decoding). On every v1 circuit the Johnson BCIKS20 figure sits below unique decoding, because its batching term grows

with the square of the evaluation domain divided by the size of a field of about 2^64 elements. [K67] For C7, the strongest theorem figure, 41.99 (unique decoding), guarantees a forger just under 2^42 hash evaluations of work and nothing beyond that. The verifier's own source prices 2^42 SHA-256 evaluations at about 7 minutes of one consumer graphics card running 10 billion hashes a second; that is arithmetic, not a measurement, the audit made no measurement on graphics cards, and no attack at that price is known. [K125] None of these figures is audited, and all of them are far below the 100 the project requires of v2 in the Johnson regime. [K68]

4.4 What one-element fingerprints and base-field challenges cost v1

Four costs follow from doing everything in one Goldilocks element (about 2^64 possible values).

Double spend (audit F03, critical). A note's commitment is one field element, so a birthday search finds two different notes with the same commitment in about 2^32 hash evaluations: generic collision bound 32.00 classical, 21.33 quantum. Two notes with one commitment have two different nullifiers, so one deposit can be withdrawn twice. This bounds every v1 pool circuit, whatever its soundness column says. An audit agent found a real collision in 1,666 seconds on 16 threads (2^33.59 evaluations) and had the verifier's code, run locally, accept a C7 spend proof for each note; no pool instruction, and so no withdrawal, was run. A second audit agent found a collision in 1,429 seconds. [K18, K19, K21]

Forgery by grinding (audit F52, high). In plain words: the verifier checks the proof at one random point, but there are too few possible points, so a forger can prepare false answers at a few thousand points and re-roll until the check lands on one of them. In detail: the out-of-domain point z is drawn from the base field and depends on the quotient commitment, while the constraint-combination challenges do not. A forger fixes the answers it wants at 4,096 chosen points and re-rolls the quotient commitment until z lands on one of them: about 2^51 draws of about four SHA-256 compressions each. The audit measured 1.1 to 1.4 × 10^8 draws per second on 24 threads of the audit's desktop, so the search takes about 186 to 236 days there, and it splits across machines. The mechanism was demonstrated on C0 (accepted at a chosen point, rejected elsewhere); no complete forgery was run, and the cost on graphics cards was not measured. The audit calls this the concrete form of the floor that the 64-bit challenge field sets. [K20]

Challenge sampling. v1 turns a hash into a challenge as an 8-byte integer reduced modulo p, which can double a field-bound error term; the "as shipped" figures above already include it. [K69]

Blinding. Apart from F69 (below), what is meant to stop a circuit-7 spend from being matched to its deposit is a blinding value with 2^63 possible values: an expected search of 62.00 (classical, log2 of hash pairs) or 31.50 (Grover). Computed, not measured (audit F66). [K28]

Two further proof-system findings are open. The transition and boundary constraint batches both start at the fixed coefficient 1, so one pair of constraints is not separated by randomness; on C7 that boundary constraint is the nullifier, and its exploitability was deliberately not measured (audit F02). [K70]

In plain words: our written argument that a proof reveals nothing about your note has a gap; no leak has been shown, and what we measured is partial evidence, not a proof. In detail, the written argument that proofs leak nothing about the note does not hold as written: every proof publishes next-row openings that the argument treats as unpublished, so its simulator can be told apart from a real proof with public data. No leak of the witness (the note's secrets) has been shown; we make no hiding claim until a corrected simulator is run (audit F69). [K71] What *is* measured, on all eight circuits: the proof's free claims (the values a simulator would draw at random) are jointly uniform in the random mask, and the trace-table entries a proof does not open stay uniform given every entry it does open. Each opened value, taken on its own, is measured uniform (the committed quotient values on all eight circuits; the trace values and the FRI layers on C7 only). Whether the opened values together can be simulated without the note is what F69 concerns, and that is not established. The solver that recovered secrets from proofs before masking was added finds the masked proofs under-determined. [K52]

Masking does not act on proofs made before it, and those proofs stay on chain. A C0 proof made before the verifier's redeploy of 2026-09-12 gives up the subscriber's secret by plain interpolation, which would let anyone make fresh valid C0 proofs for that subscription, beyond the replay of section 8; users made C0 proofs before that date (section 3.6). A C1 proof made before 2026-08-31 gives up its private inputs to a solver that knows the circuit's rules (its public inputs already named the note's commitment). Both are shown by tests on the old proof shapes; how many such proofs are on chain has not been counted. [K147]

---

5. Known limits of v1

We lead with the two findings that break the pool for everyone and that only v2 can fix, stated plainly. Both are known and both are devnet-only. Several of the findings below also lose users' or the operator's money (F05, F11, F27, F56, F70). [K72]

1. One deposit can be withdrawn twice (F03, critical). Anyone with a desktop computer and less than half an hour can find two notes that share one fingerprint; depositing once, they could then withdraw twice. An audit agent found such a pair in under 28 minutes and had the verifier's code, run locally, accept a spend proof for each; no withdrawal was run. [K18, K19] v2 is designed to widen the fingerprint to four field elements (section 8). [K22]

2. Any statement can be forged by grinding (F52, high). About 2^51 attempts, about 186 to 236 days on 24 threads of the audit's desktop, or about one day on some 187 to 237 such desktops, would produce a proof the verifier accepts for a false statement. For a spend, that

means SOL withdrawn that was never deposited (our reading; no complete forgery was run). [K20] v2 is designed to draw every challenge from a cubic extension field (section 8). [K23]

The other high findings, in plain words, as the audit found them at commit `4b122265`. Web changes that fix or mitigate F07, F11, F15, F56, F70 and F72 went live on 2026-09-23, and the web client now stops before any proof or payment when it detects the states of F01 and F28 and refuses the C1+C3 path of F05 by default; section 6.4 gives the status of each. No finding that needs a program change is fixed: those rows still describe the programs that run on devnet today, and `p01_liquidity` (F27) has been set inactive instead. [K72, K85, K130, K131]

ID	WHAT THE AUDIT FOUND	NEEDS
F01 (with the related F06, medium)	A depositor can write their note into any empty slot of the insertion subtree; clients that rebuild the tree from events then disagree with the chain and get stuck. No theft.	circuit change and redeploy
F02	Two groups of checks are combined without a random weight on one pair, so an error in one could in principle offset the other; on C7 one of those checks is the double-spend tag. Whether this can be exploited was not measured (section 4.4).	prover and verifier change
F05	A spend on the older C1+C3 path does not bind the payee, so someone who copies public proof bytes can redirect the withdrawal; shown in a local test with real proofs. C7 is not affected.	redeploy; meanwhile drop the older C1+C3 path from clients
F07	After the default Shield, the only copy of the issued note is a browser storage entry, and the screen says there is nothing to back up.	web fix
F11	The relay route can pay the float back to a caller-chosen address while the claim route still issues a note for the same payment: a net gain of 0.64 SOL per cycle, plus a note.	web fix
F15	Public text promises "no double spending" while F03 is known.	text fix
F27	The deployed `p01_liquidity` program pays a caller-chosen amount against a proof that has not finished verification; an audit agent drained its devnet reserve in a local test. No shipping flow calls it. Its pool was set inactive on devnet on 2026-09-22 (section 6.4).	disable or redesign
F28	One deposit carrying a value written in a form the program should have rejected (non-canonical) permanently blocks all later deposits and transfers of that pool (v4 withdrawals still work), for about 0.007 SOL.	redeploy
F52	Forgery by grinding (above).	v2

ID	WHAT THE AUDIT FOUND	NEEDS
F56	The "Deposit my own note (linked to my wallet)" button pays the till and then refuses to relay, losing the buyer about 1.013 SOL per click.	web fix
F70	In the note-in exchange, whoever pays the fee of the withdrawal to the till gets the claim code, and a copier of the proof can become that payer.	web fix, or bind the submitter in C7 (redeploy)
F72	A stranger whose request names the same position in the pool's tree can take the claim of a successful contribution; the partial fix of the audit's close left this possible when the deposit lands more than 20 minutes after the relay (the mitigation live since 2026-09-23 moves that bound to at most 3 hours, section 6.4).	web fix

Every finding, with its file and its evidence, is in the audit findings file, and the medium and low ones are summarised in section 6. [K72]

Other limits that are not audit findings. The anonymity crowd is small (residual 4). [K13] The mobile app's published build carries a prover older than the current verifier, so its proofs are rejected until a new build ships. [K73] The shipped prover has been rebuilt byte for byte only on a Windows x86-64 host; a Linux rebuild of the previous blob differed (section 9). [K74]

---

6. The internal audit

6.1 What it is, and what it is not

It is an **internal** audit run by AI agents (instances of Claude Opus 5.5) in four roles: researchers, skeptics, fixers and verifying agents. It is **not** an external audit by humans and does not stand in for one; no independent human has reviewed it. It covers v1 as it runs on devnet, at commit `4b122265`, and not v2. [K3, K75] The repository first moved to `7a007e2c`, two commits that change only test budgets and a lint directive (3 test files). [K127] On 2026-09-23 it moved again, to `c0865f89` (`origin/master` as of the writing of this edition, before the commit that publishes it): the audit's fixes, the relicensing, the benchmark harness, the removal of unused code (section 3.6), the prover without C2, C4 and C5, and a rewritten README. The audit did not re-run on that code. The commit that publishes this edition adds the paper, its page and its PDF, the benchmark results of section 7, and small changes, among them: the dead C2, C4 and C5 constants of the web and extension clients, the three tests that ran Groth16 proofs with `snarkjs` and the root `snarkjs`, `circomlib` and `circomlibjs` dependencies, links to the paper from the site's footer and documentation pages, a corrected compute-unit record of the 2026-09-12 acceptance in `packages/stark-prover/deployed-verifier.json` (its 889,570 CU were phase 1 only), the benchmark harness's OS-width field, and the lists of the security-levels prose gate. The audit re-examined none of it. [K135, K60]

6.2 Method

- **Eight axes:** the circuit constraints; the FRI and Fiat-Shamir parameters; the hash; the verifier program; the pool program; the web client and prover loader; the server, key-value store, API and CI; and the public claims. The browser extension and the mobile app were out of scope. [K76]
- **Fresh context.** Each researcher started from the code, without earlier reports. The rule was broken three times, each declared by the agent, and no confirmed finding depends on those reads. [K77]
- **Three skeptics per finding, majority vote.** A finding is confirmed only with at least two of three votes; the skeptics re-ran the probes themselves when they could. [K78]
- **Red before green.** Every fix starts with a test that fails on the old code and passes after the fix, and a separate verifying agent restores the old code byte for byte to watch the test go red again. [K79]
- **Stop rule, not reached.** The audit was to loop until two rounds in a row found nothing new. Rounds 1 to 4 confirmed 24, 27, 16 and 12 findings, so the audit stopped **unfinished**; another round would likely find more, especially in the server's money routes and the client. [K80]
- **No transactions.** No deploy, no transaction on any cluster, no production request; devnet reads were read-only. Exploits ran in a local Solana VM (litesvm) against programs built from the repository, or in the web test runner against the real routes with fake RPC and storage. [K81]

6.3 What it covered and did not

Covered, among other things: the constraints of the live circuits, compared one by one with the verifier's on-chain evaluators for C0, C1, C3, C6 and C7; every v1 parameter against the calculator; the transcript order on both sides, with 811,490 mutations of real proofs, each flipping a single bit, over four probe runs, none of them accepted; bit-for-bit agreement of the Poseidon implementations over more than a million inputs, the shipped prover included; every instruction of the verifier and the pool; the 19 API routes and 11 workflows. [K82]

Not covered, or covered less than the list above suggests: no end-to-end transaction on devnet; no comparison between the deployed binaries and the source; no complete forgery run through DEEP-ALI and FRI; no measurement on graphics cards; no cryptanalysis of the Poseidon constants beyond a structural check; the theorems the calculator cites (BCIKS20, BCHKS25) were not re-derived, and the v1 and v2 figures in the theorem regimes rest on them; no real browser; the production configuration of Vercel and of the GitHub repository. C2, C4 and C5 were only skimmed, since no deployed instruction consumes them. Several pool probes planted "verified" proof buffers directly into the local chain state, so they tested the pool's logic without going through the verifier. The machine's system disk filled during rounds 2 to 4, so some skeptics could not re-run their probe and voted from reading the code, which their votes record. [K83]

6.4 Results

Findings in total: 116 raw, 94 new after removing duplicates, 79 confirmed, 15 rejected. By severity, the 79 confirmed findings are 1 critical, 12 high, 16 medium, 48 low and 2 informational. [K84]

The block below is the reference for the fix status of the findings. It is updated as fixes land, are committed and are deployed, and the other places in this paper that state a fix status (listed in `docs/CLAIMS.md`, section "Where fix status and benchmark figures are stated") are updated with it.

Audit status (as of the writing of this edition on 2026-09-23, before the commit that publishes it: code at `origin/master` `c0865f89`, production deployment `dpl_2GAWddKM…` of that commit)

The counts are the audit's own, recorded at its close on 2026-09-22, when every fix was still uncommitted. They were not re-tallied after the commits of 2026-09-23, and the audit's two-dry-rounds stop rule was never reached (section 6.2), so they describe what the audit found and fixed, not a finished review. [K80, K84, K85]

STATUS AT THE AUDIT'S CLOSE (2026-09-22)	MEANING	COUNT
Fixed	fix made, test red then green, confirmed by a verifying agent	9
Partly fixed	part fixed; the rest was outside what the fixing agents were allowed to change (often a translation or a generated PDF)	33
Needs the founder	needs an on-chain change, a redeploy or a decision	17
Open	no fix yet	20
Total confirmed		**79**

Committed on 2026-09-23: the audit's fixes, in six commits (`a0419883`, `9f3b178e`, `87180c55`, `25cf3444`, `42d454e9`, `b7e20cd1`), then the items the commit gate found still blocking a push, in `065412d0` (among them F11 and F56). **Deployed**: pushed to `master`; the web part has been in production since deployment `dpl_4cKkKP1p…` of `065412d0` (2026-09-23 02:46 UTC), and at the writing of this edition, before the commit that publishes it, production was `dpl_2GAWddKM…` of `c0865f89` (03:27 UTC). No post-deploy probe of these fixes is recorded. **On chain**: no program was changed or redeployed; the pool of `p01_liquidity` was set inactive (F27).

CRITICAL AND HIGH FINDINGS	STATUS ON 2026-09-23
F03 double spend	open on chain; needs v2
F01 insertion position (F06, medium, is the same issue)	open on chain; needs a circuit change and a redeploy. Mitigation live since 2026-09-23: the web client stops before any proof or payment when the tree it rebuilds disagrees with the chain
F02 fixed coefficient	open on chain; needs a prover and verifier change
F05 C1+C3 payee not bound	open on chain; needs a redeploy. Mitigation live since 2026-09-23: the web app no longer sends C1+C3 spends unless a build flag, not set in production, turns them back on; the program still accepts them, and the extension's transfer still uses them
F07 issued note only in browser storage	partly fixed at the audit's close, when the SubscribePanel and resume parts were still open; further parts (the only-copy warning and the recovery code in English and French) live since 2026-09-23; the SubscribePanel and resume parts have not been re-checked since; not re-examined by the audit, not probed after the deploy
F11 relay double payout	fixed in the web app (test red then green), live since 2026-09-23; not re-examined by the audit, not probed after the deploy
F15 "no double spending" text	English text and README fixed at the audit's close, French text on 2026-09-23; live since 2026-09-23; not re-examined by the audit, not probed after the deploy
F27 `p01_liquidity` prefund	mitigated: its pool was set inactive on devnet on 2026-09-22 (`update_params`, signature `5ZLniWNV…`, slot 502,627,961), and `prefund` requires an active pool; the repository's SDK no longer contains the module that drove it (deleted 2026-09-23); privacy-sdk versions published earlier still do. The program is still deployed and its pool still holds 569,708,499 lamports; it needs a redesign or a close
F28 non-canonical deposit	open on chain; needs a redeploy. Mitigation live since 2026-09-23: the web client stops before any proof or payment when the pool is in the blocked state
F52 forgery by grinding	open on chain; needs v2
F56 own deposit pays and refuses	fixed in the web app (test red then green), live since 2026-09-23; not re-examined by the audit, not probed after the deploy
F70 note-in exchange claim capture	mitigated, not fixed: the exchange is switched off in the web client and in the claim route, live since 2026-09-23 (section 3.2). A caller that skips the client's refusal still lands its withdrawal to the till and gets no note (0.995 SOL lost each time; three such withdrawals in our benchmark trial, section 7.2)

CRITICAL AND HIGH FINDINGS	STATUS ON 2026-09-23
F72 contribution claim capture	mitigated, not fixed: a hold lasts while its deposit can still land, at most 3 hours by default; past that the takeover is still possible; live since 2026-09-23

Among the medium and low findings, the commit messages of 2026-09-23 describe changes, as fixes, mitigations or partial fixes, for F04, F09, F13, F14, F16 to F24, F26, F33, F35 to F51, F53, F57 to F66, F68, F71 and F73 to F78; that list is the commits' own account, not a re-examination by the audit. By the same account some remain open or partial: the server-side RPC key of F36 and its rotation, one workflow of F43, a residual of F62 (a payment one hop further from the float) and F74. F69 is not fixed: the README's hiding claim was withdrawn and a test pins the published openings (`42d454e9`); the simulation argument is not corrected (section 4.4). [K141, K71]

Source of this block: `AUDIT-V1-FINDINGS.json` (`totals`, `statusLegend`, `findings[].status`, as of the audit's close on 2026-09-22); the commit messages of `a0419883` to `b7e20cd1` and `065412d0`; the production deployments read on 2026-09-23; and read-only devnet reads of 2026-09-23 for F27. [K85, K130, K131, K134, K140, K142]

No finding could be fixed on chain during the audit, because it forbade redeploys. The audit's fixes are in the web app, the SDKs, CI and the documentation. They were in the working tree only at the audit's close (2026-09-22); they were committed and pushed on 2026-09-23, and the web part is live in production since then (the block above is kept current). A fix counts for users only once it is deployed, and, for a leak, probed after the deploy; no such probe is recorded yet. (The leak-ledger fixes of sections 2 and 3 are a separate, earlier set: committed on 2026-09-20, part of the audited code, and live since 2026-09-22.) [K85, K120] The medium and low findings cover, in groups: pool-program edge cases that need a redeploy (fee accounting on token pools, era rollover, the root ring after a migration, rent taken from notes); web client storage and recovery; server rate limits, locks and admin login; public text that promised more than the code (at the audit's close, several were corrected in English while the French text and some generated PDFs were not); and the project's own SDK packages under `packages/` (`@protocol-01` specter-sdk, privacy-sdk, privacy-toolkit, p01-js and auth-sdk), where the audit found code that did not do what the README said (privacy-toolkit, and the dead modules of privacy-sdk, p01-js and specter-sdk, were deleted on 2026-09-23). [K86, K133]

6.5 What comes next

An **external review** by independent cryptographers and Solana auditors, starting from this audit and from what it did not cover (6.3). The audit is their starting point, not a conclusion. [K24]

7. Performance

7.1 Method

We measure with the method written in `docs/BENCHMARK-METHOD.md` **before** the measurement it describes; if a result and the method disagree, the result is wrong. [K87] The main rules:

- **What.** v1 as deployed, a pre-v2 baseline: the verifier `DGY37k3J…`, the shipped prover `241caaab…`, v3 deposits, v4 spends, on devnet only. Nothing here is a mainnet figure, and nothing here predicts v2. [K88]
- **Flows.** Proving alone (native Rust, meaning compiled code running directly on the CPU; the shipped WebAssembly in Node, and the same in a browser worker); the proof pipeline on devnet (prove, upload, verify, and close, meaning closing the proof account and getting its rent back); and four product flows end to end through the app's own code: a deposit from your wallet, a withdrawal, a subscription, and the note-in exchange (which the method calls `purchase`). The time to fund the one-time key is inside every product time, because a user waits for it. [K89]
- **Cold and warm.** A *cold cache* is a browser with no saved pool history, like a new visitor; *warm* is a returning user. For the proving flows, cold means a fresh process or worker that must load the prover. [K89]
- **Samples.** At least 30 per flow, per circuit and per cold or warm mode; every sample is a fresh proof; failures are counted, not dropped. A run with fewer than 30 good samples, on a busy machine (above 15% CPU), on an unaccepted Node version or without its RPC plan recorded is marked not publishable, and no time from it is given here as a result: 7.2 says only what ran, its n, and why it is not a result. The single runs of 7.2 (one or two runs each, K136) are quoted only as labelled single runs, never as results. [K90, K136]
- **Reported.** n (the number of samples), minimum, median, p90 (the time that 9 runs in 10 beat; nearest rank, always an observed value), maximum, spread (maximum minus minimum) and max/min (maximum divided by minimum). [K90]
- **Machine.** One desktop: Intel Core i9-14900K (24 cores, 32 threads), 48 GiB DDR5, Windows 11 Pro 25H2 with the AtlasOS playbook and its power plan. A figure from this machine is a figure for this machine only. [K91]
- **Records.** Every run writes a manifest (machine, commit, blob digest, programs, deploy slots), raw logs scrubbed of secrets, and every transaction signature in full, under `docs/bench/<date>/<run>/`. The measurement key is used for nothing else, so its deposits and spends are linkable by design; they say nothing about users. [K92]

Earlier figures (18.6 s to shield and 20.8 s to withdraw, measured on 2026-09-12) were single runs made with the previous prover and left the funding step out of two of the three flows; the method lists what they got wrong, and we do not use them. [K93] The figures of 2026-09-23 in 7.2 under "Single runs" are one or two runs each and are not publishable

under the method either; they are kept to show where the time goes. For the withdrawal, the benchmark row of 7.2 now replaces them; for the deposit and the subscription, no publishable benchmark exists yet (7.2 says why). [K136]

7.2 Results

Four runs of 2026-09-23 feed this section, all under the method of 7.1, all on the same machine, and every devnet run on the RPC plan its manifest records: the Helius Free plan, with the app's public key. The proving tables come from the local run `run-20260923T031450Z` (below). The proof pipeline comes from the devnet run `run-20260923T032635Z` (03:26 to 04:28 UTC, commit `2221556c`, Node 24.21.0, one modified tracked file, verdict **publishable**). The withdrawal comes from the devnet run `run-20260923T042841Z` (04:28 to 05:13 UTC, commit `c0865f89`, Node 24.21.0, verdict **publishable**); its tree had 21 modified tracked files, the uncommitted edits of that morning, which its manifest records. The deposit comes from the devnet run `run-20260923T051434Z`, which is **not publishable** (below). Times in seconds unless marked ms. Every figure and every signature is in `docs/bench/2026-09-23/<run>/`; spread and max/min are in each run's summary.md. [K143, K144, K145]

Product flows, devnet, end to end.

FLOW	N	MEDIAN	P90	MIN	MAX	PUBLISHABLE
Withdrawal, warm cache (run `run-20260923T042841Z`)	30	32.3	36.2	28.9	38.6	yes (0 failed)
Withdrawal, cold cache	not measured	—	—	—	—	no run
Deposit 1 SOL from your wallet (own-deposit path, C6; run `run-20260923T051434Z`)	28 of 30	not a result	—	—	—	no: 2 samples failed
Subscription, warm cache	3 (a trial run)	not a result	—	—	—	no: 3 samples, the method needs 30
Subscription, cold cache	not measured	—	—	—	—	no run
Note-in exchange (the method's `purchase` flow), warm cache	0 of 2, plus a warm-up	not measured	—	—	—	no: refused in production

FLOW	N	MEDIAN	P90	MIN	MAX	PUBLISHABLE
Note-in exchange (the method's `purchase` flow), cold cache	not measured	—	—	—	—	no run

The withdrawal ran the app's own code in a test harness on our machine (a Node test process, not a browser: no interface and no wallet prompts), with the one-time key funded directly by the measurement wallet rather than the operator's float of the default path, and the pool's history already cached. It is timed from the moment the harness has derived the payee to the moment the withdrawal lands at `confirmed` commitment; funding the one-time key is inside that time; the scan of the pool's history, and the deposit a sample makes first when its key holds no unspent note, are timed apart and not counted. [K143]

What ran and is not a result. The method needs 30 good samples per row and counts every failure (7.1), so none of the following gives a time.

- *Deposit.* Run `run-20260923T051434Z` (05:14 to 05:33 UTC) completed 28 of its 30 samples. Two stopped during the upload of the proof, before the deposit instruction was sent, when the harness's test-runner process died (in one, the channel between the runner and its worker closed; in the other, the process exited abnormally). With 28 good samples the run is not publishable, and we quote none of its times. [K145]
- *Subscription.* Only a trial run of 3 samples (plus a warm-up) was made, because the measurement key's remaining devnet SOL allowed only 3 (each sample first deposits a 1 SOL note of its own). The harness refuses to write a run of fewer than 30 samples under `docs/bench/`, so this one is kept in the benchmark's working notes, not in the repository, and it is not a result. [K145]
- *Note-in exchange.* Not measured. A trial of 2 samples plus a warm-up against the production deployment (styx.cash) stopped at the claim step in all three: production refuses the claim with `EXCHANGE_DISABLED` ("the note-in exchange is disabled on this deployment"): the exchange has been switched off there since 2026-09-23 (audit finding F70, section 3.2). The harness sends the withdrawal to the till before it asks for the claim, and does not go through the web client's refusal, so its three withdrawals to the till landed on devnet and got no note: 3 × 0.995 devnet SOL paid to the till. The claim route's own answer says that only a withdrawal that landed before the switch-off is settled through support. [K145, K131]

Waiting on the RPC's rate limit. The Helius Free plan answers a burst of requests with HTTP 429 ("too many requests"), and the client waits and sends again. A request observer ran with the devnet runs and recorded every RPC request and its answer. We count as a *stall* the time from sending a request that was refused to sending the next request that was answered (the refused request's round trip plus the client's back-off), with overlapping waits counted once. Over the whole withdrawal run, warm-up, deposit legs, scans and probes

included, 1,324 of 5,599 RPC requests were refused. Within the 30 timed withdrawals, every sample waited, and the stall was 63.2% of their summed product time; per sample, the stall had a median of 20.5 s, a p90 of 25.0 s and a maximum of 25.3 s. That wait is set by the RPC plan: a limit of the RPC plan the app uses today (styx.cash ships with the same Free-plan key, audit F36), which users of the live app also wait on, not work done by Styx's code. That is why the method records the plan with every run. For this run we give no withdrawal time "without the wait": a subtraction is not a measurement (the single runs below give one, labelled as such), and no run was made on a plan without that limit. The proof pipeline run was observed as well (2,721 of 8,382 requests refused), but the observer could not split its stall by sample, so its share is not stated; its times below include those waits. WebSocket refusals were not captured. [K146]

Proving time, this machine, no network (ms). Local run `run-20260923T031450Z` of 2026-09-23 (03:14 to 03:22 UTC), commit `2221556c`, Node 24.21.0, prover `241caaab…` (240,172 bytes), 30 samples per row and 30 per cold row, verdict **publishable**: every flow passed the checks of the method (CPU 10.1% busy before the run; 7.5%, 12.4% and 9.6% on average during the native, Node and browser flows, against the 15% limit; one sample of the whole run peaked at 87.0%). The run's tree had one modified tracked file, recorded in its manifest. Browser rows are what a user's tab does, measured in headless Chrome 154 on the same machine, in a module worker. Cold includes loading the prover (a new Node process, or a new browser worker, per sample); the browser's download of the prover from a local server is recorded apart and not counted. The method does not separate cold and warm for native proving. Every figure is in `docs/bench/2026-09-23/run-20260923T031450Z/`, with the minimum, maximum, spread and max/min of each row in its summary.md. [K137, K138]

CIRCUIT	NATIVE MEDIAN / P90	NODE WARM MEDIAN / P90	NODE COLD MEDIAN / P90	BROWSER WARM MEDIAN / P90	BROWSER COLD MEDIAN / P90	N (BROWSER WARM)
C6 (deposit)	120 / 267	578 / 1,210	495 / 1,266	292 / 845	478 / 1,095	30
C7 (spend)	70 / 277	727 / 2,199	343 / 930	488 / 1,297	457 / 1,765	30
C1 (older spend)	95 / 328	434 / 956	423 / 1,867	324 / 968	318 / 1,135	30
C3 (older spend)	130 / 379	383 / 793	650 / 1,576	310 / 1,388	405 / 952	30
C0 (pause, resume)	161 / 467	307 / 1,050	405 / 1,812	335 / 915	625 / 1,096	30

How to read it. The spread is wide: within a row the slowest proof took 19.9 to 152.9 times as long as the fastest (native C7: 27 ms to 719 ms). Our reading, not a separate measurement, is that most of it is the proof-of-work step, whose number of attempts changes from one proof to the next; the harness does not time that step apart. With 30 samples, that variation is larger than the cost of loading the prover, so a cold median can come out below the warm median of the same circuit (Node C7: 343 ms cold, 727 ms warm); the cold rows do not show that loading is free. [K138]

Proof pipeline on devnet: prove, upload, verify, close (seconds). Run `run-20260923T032635Z`, 30 samples per circuit, 0 failed on every circuit. One sample is a fresh proof from the shipped WebAssembly prover in Node (a warm process: one proof per circuit made and discarded first), then the allocation of a proof account, the upload of the proof in 4,096-byte transactions, its verification (phases 1 and 2) and the closing of the account, each step waited for at `confirmed` commitment. The pool instruction that consumes the verified proof is not timed. These times include the RPC's rate-limit waits (above). [K144]

CIRCUIT	N	MEDIAN	P90	MIN	MAX
C6	30	26.8	28.5	22.1	50.5
C7	30	22.1	27.3	16.7	31.0
C1	30	28.2	29.2	20.7	32.4
C3	30	22.3	26.0	17.5	27.3
C0	30	22.2	24.5	14.5	50.4

The method measured C2, C4 and C5 as well until 2026-09-23. Since the shipped prover no longer exports them (section 3.6), the harness runs the five live circuits only, and the local run above has no rows for them. [K59, K137]

Single runs, devnet, 2026-09-23 (not a benchmark). Before the benchmark's product flows, the changes of commit `065412d0` were timed on devnet with a harness that calls the web app's own code in a Node process (not a browser), against the Helius Free-plan RPC, with a key made for measurement. One or two runs per flow, so none of this is publishable under the method, and none of it replaces the table of product flows above. [K136]

FLOW, AS TIMED	RUNS	END TO END	OF WHICH WAITING ON THE RPC'S RATE LIMIT	WITHOUT THAT WAIT (SUBTRACTION, NOT A RUN)
Deposit of 1 SOL by the measurement key through a one-time key it funds (the path the key can drive alone; not the default Shield path of section 3.2), prover `d5583d41…`	2	36.6 s (37.1 s and 36.0 s), against 63.9 s for the same deposit before the changes	23.1 s (median of the two)	13.5 s, against 40.2 s before
Withdrawal, click to landed, clicked after the page-load history scan had finished (scan not included), one-time key funded from the wallet (path C), prover `241caaab…`	1	32.0 s (33.3 s to the end of execute, with close and sweep)	20.7 s	about 11.3 s
Subscription, click to landed, clicked after the page-load history scan had finished (scan not included), one-time key funded from the wallet (path C), prover `241caaab…`	1	37.5 s (40.8 s to the end of execute, with close and sweep)	27.5 s	about 10.0 s

Three conditions hold for the withdrawal and subscription rows. The harness started with an empty history store and waited for the page-load scan of the pool's history to finish (45.3 s and 45.5 s) before it clicked, so 32.0 s and 37.5 s do not include the history walk that a visit with an empty history store makes; they are not the time a new visitor waits. The one-time key was funded from the measurement wallet directly (path C), not from the operator's float as in the app's default path. And the quoted time stops when the spend landed: the close and sweep that follow bring click to end of execute to 33.3 s and 40.8 s. [K136]

The wait is the Free plan's limit on `sendTransaction` (HTTP 429 answers and the client's back-off), not Styx's own work; it depends on the RPC plan, which is why the benchmark records the plan it ran on. The proof itself took 0.81 s (withdrawal) and 0.29 s (subscription) of those times. [K136]

What the proving figures cover. The prover moved from point-by-point evaluation to a number-theoretic transform (a faster method for the prover's polynomial arithmetic) before this run: outside the proof-of-work step, native proving became at least 23 to 33 times faster on the five live circuits, on this machine (an internal measurement of 2026-09-20, n = 10 per circuit, not a benchmark under the method of 7.1), with byte-identical proofs, and the proof-of-work (about 2^22 hash attempts on average, some 169 to 173 ms natively) is now most of the native time. [K94] What this benchmark does not show: v2, mainnet, phones or laptops,

the relay path, the interface, the default Shield path (pay the till, redeem a claim code), which is not measured as one flow, and anything about privacy or soundness. [K95]

7.3 Comparison with other systems, on two axes

We compare on **time** and on **trust model** together, and we rank no one on time alone. Apart from the Styx runs in 7.2, no system in this table publishes an end-to-end latency measured under a written method with its hardware, so no "X times faster" statement can be made. [K96] Every figure and phrase below is quoted from its source and carries the source's id in brackets; the list after the tables gives, for each id, the full URL, the date shown on the page and the kind of source. Every rival quote was checked against a saved copy of its page on 2026-09-22, except one, marked "read through a web reader only", whose server refused a direct download. [K97]

Time.

SYSTEM	FIGURE	KIND OF SOURCE	SOURCE
Umbra (on Arcium; MPC, a committee of computers that compute on encrypted data)	"a few seconds longer than a standard token transfer"	published by the vendor	[umbra-latency]
Zcash (Orchard)	block time "drops from 75 seconds to 25" with NU7, "targeting Nov. 5"	third-party report (press)	[zcash-nu7-blocks]
Zcash	transaction construction "from more than three seconds to below 200 milliseconds in some benchmark tests"	developer benchmark (published by a developer, not independently reproduced), reported by the press	[zcash-zakura]
MagicBlock (TEE, a sealed hardware enclave you must trust)	"Sub-50ms private execution", settlement on Solana not counted	published by the vendor	[mb-latency]
NEAR Confidential Intents	"identical speed and cost", which the integrator itself calls "NEAR's claim"	published by the vendor, reported by an integrator	[near-speed-claim]

SYSTEM	FIGURE	KIND OF SOURCE	SOURCE
Sombra (Bonsol Labs)	"~1.8-2 s proof (RTX 3090)"; "settles in 412 ms", with no hardware, conditions or method	published by the vendor	[sombra-proof-time], [sombra-settle]
Styx v1	the tables of 7.2	measured by us, under the method of 7.1	`docs/bench/2026-09-23/run-20260923T031450Z/` (proving), `run-20260923T032635Z/` (pipeline) and `run-20260923T042841Z/` (withdrawal), in the same folder

Trust model.

SYSTEM	WHOSE FAILURE BREAKS PRIVACY	WHOSE FAILURE BREAKS FUNDS	EXPOSED TO A QUANTUM COMPUTER	WHERE THE PROOF OR ATTESTATION IS CHECKED	CODE
Umbra / Arcium (MPC: a committee of computers that compute on encrypted data)	the MPC cluster: Umbra writes "compromising a majority of them simultaneously" [umbra-trust-majority]; Arcium writes privacy holds if "at least one Cluster member is honest" [arcium-trust-one-honest] (the two texts differ); network "permissioned", started with a "trusted dealer" [arcium-mainnet-alpha]	the cluster, and the setup of the Groth16 note proofs (Groth16: a curve-based proof system that needs a setup ceremony) [umbra-groth16], of which no ceremony is mentioned in the SDK docs [umbra-ceremony] (a forged setup lets proofs be forged, a general property of Groth16)*	yes*, from X25519 keys [arcium-x25519] and Groth16 over BN254 (the elliptic curve those proofs use) [umbra-bn254]; neither Umbra nor Arcium states anything on quantum	note proofs: "Groth16 ZK proof verification on burn" [umbra-groth16]; MPC: off chain, by the cluster	program and circuits not found in its public repositories [umbra-source]

SYSTEM	WHOSE FAILURE BREAKS PRIVACY	WHOSE FAILURE BREAKS FUNDS	EXPOSED TO A QUANTUM COMPUTER	WHERE THE PROOF OR ATTESTATION IS CHECKED	CODE
Zcash (Orchard)	no party beyond the chain and its assumptions* (no trusted setup since 2022 [zcash-no-setup])	same*	yes, stated by Zcash in a proposal (ZIP 2005, status Proposed): an adversary able to find discrete logarithms "would be able to steal or forge funds" [zcash-quantum-funds]	by the chain's consensus*	public repositories, licence of each not re-read [zcash-source]
MagicBlock (TEE: a sealed hardware enclave you must trust, here Intel TDX)	Intel hardware and its attestation ("Trust assumption in vendor hardware" [mb-trust-vendor]); two published physical attacks break TDX attestation or integrity [tdx-teefail] [tdx-ddrop], and Intel calls such physical attacks out of scope [tdx-intel-scope] (read through a web reader only)	the enclave and its operator*	yes: P-256 attestation key [tdx-p256] and Ed25519	by a hardware attestation; there is no proof*	validator under Business Source License 1.1 (source-available); no complete audit of the validator [mb-source]

SYSTEM	WHOSE FAILURE BREAKS PRIVACY	WHOSE FAILURE BREAKS FUNDS	EXPOSED TO A QUANTUM COMPUTER	WHERE THE PROOF OR ATTESTATION IS CHECKED	CODE
NEAR Confidential Intents	permissioned validators of a private shard [near-shard], and a TEE bridge. LeoDex, which ships the product in its own app, writes that this means trusting "a validator set whose composition is not fully public" and that "the attestation details have not been published" [near-trust-leodex]	same	the bridge and classical signatures, yes; ML-DSA-65 accounts are available [near-mldsa]	on the private shard; no client-side proof	Intents contracts public under MIT; private shard code not found [near-source]
Sombra	not fully stated: an "attestation quorum" of nodes "must independently verify the ZK proof" [sombra-verify-site]; the deposit link is "observable and … permanent" by their own white paper [sombra-deposit-link]	not stated: quorum size and members not published [sombra-verify-wp]	not for the proof, by their own account [sombra-crypto]; unknown if a Groth16 wrapper is used on chain [bonsol-groth16]	"submitted and verified" on Solana L1 and a node quorum [sombra-verify-site]; the exact on-chain mechanism is not specified	no code or licence linked [sombra-source]

SYSTEM	WHOSE FAILURE BREAKS PRIVACY	WHOSE FAILURE BREAKS FUNDS	EXPOSED TO A QUANTUM COMPUTER	WHERE THE PROOF OR ATTESTATION IS CHECKED	CODE
Styx v1	the operator for notes it issues (it can recompute them) and for its off-chain map; Helius for IPs; among the links still open in production (ledger B11 for every client; D2 and E2 to E4; E5, the operator's stock can be told apart; B12 and C7 for the extension and mobile app; D3 for the extension; sections 2 and 3.3); beyond those, the chain and the hash assumptions, within section 4, whose argument that a proof hides the note has a gap (F69)	the operator, which can spend every issued note until its holder spends it; the soundness of section 4 and the open findings of section 5, F03 and F52 above all; the upgrade authority	spend authority (Ed25519) yes; key derivation from Ed25519 yes; the proof rests on no curve, so Shor's algorithm does not apply, but v1's quantum and classical floors are low and F03 and F52 already break v1 classically (section 4)	inside a Solana program, in full	private repository since 2026-10-02 (section 9.5); internal AI-assisted audit only

* inference, not stated by the source. [K98, K99]

Sources for 7.3. Each line: id, URL, date shown on the page, kind of source. All pages were read on 2026-09-22. [K98, K99, K100]

- [umbra-latency] https://sdk.umbraprivacy.com/llms-full.txt, no date on page, published by the vendor.
- [zcash-nu7-blocks] https://decrypt.co/378667/zcash-upgrade-make-private-payments-3x-faster, 2026-09-18, third-party report (no measurement of its own).
- [zcash-zakura] https://crypto.news/zcash-private-transactions-could-fall-below-200ms/, 2026-08-31, developer benchmark (published by a developer, not independently reproduced).
- [mb-latency] https://www.prnewswire.com/news-releases/magicblock-brings-institutional-grade-privacy-to-solana-in-industry-first-302543835.html, 2025-09-02,

published by the vendor (press release).

- [near-speed-claim] https://leodex.io/editorial/near-confidential-intents-july-2026, 2026-07-10, vendor claim reported by an integrator.
- [sombra-proof-time], [sombra-settle] https://sombra.tech/, no date on page, published by the vendor.
- [umbra-trust-majority] https://sdk.umbraprivacy.com/llms-full.txt, no date on page, published by the vendor.
- [arcium-trust-one-honest] https://docs.arcium.com/multi-party-execution-environments-mxes/mpc-protocols, no date on page, published by the vendor.
- [arcium-mainnet-alpha] https://arcium.substack.com/p/arcium-mainnet-alpha-is-live, 2026-02-04, published by the vendor.
- [umbra-groth16] https://sdk.umbraprivacy.com/introduction, no date on page, published by the vendor.
- [umbra-bn254] https://sdk.umbraprivacy.com/llms-full.txt, no date on page, published by the vendor.
- [arcium-x25519] https://docs.arcium.com/developers/encryption, no date on page, published by the vendor.
- [umbra-ceremony] https://sdk.umbraprivacy.com/llms-full.txt, no date on page: searched for and not found in the sources read.
- [umbra-source] https://github.com/umbra-defi, observed by us on 2026-09-22 (a repository listing; no vendor sentence).
- [zcash-no-setup] https://z.cash/upgrade/nu5/, activation on 2022-05-31, published by the vendor.
- [zcash-quantum-funds] https://zips.z.cash/zip-2005, created 2025-03-31, status Proposed, published by the vendor.
- [zcash-source] https://github.com/zcash, observed by us on 2026-09-22 (a repository listing; no vendor sentence; licence of each repository not re-read).
- [mb-trust-vendor] https://docs.magicblock.gg/pages/private-ephemeral-rollups-pers/introduction/onchain-privacy, no date on page, published by the vendor.
- [tdx-teefail] https://thehackernews.com/2025/10/new-teefail-side-channel-attack.html, 2025-10-28, security research (as reported by the press).
- [tdx-ddrop] https://thehackernews.com/2026/09/new-ddrop-attack-breaks-intel-tdx-and.html, 2026-09-14, security research (as reported by the press).
- [tdx-intel-scope] https://www.intel.com/content/www/us/en/security-center/announcement/intel-security-announcement-2025-10-28-001.html, 2025-10-28, vendor security advisory; read through a web reader only.
- [tdx-p256] https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/Intel_TDX_DCAP_Quoting_Library_API.pdf, no date read, published by

the vendor.

- [mb-source] https://github.com/magicblock-labs/magicblock-validator/, read 2026-09-22, published by the vendor.
- [near-shard] https://mpost.io/near-unveils-confidential-intents-to-enable-privacy-first-execution-for-cross-chain-transactions/, 2026-02-25, vendor claim reported by the press.
- [near-trust-leodex] https://leodex.io/editorial/near-confidential-intents-july-2026, 2026-07-10, integrator report (written by a party that ships the product in its own app, so not independent).
- [near-mldsa] https://docs.near.org/protocol/access-keys, no date on page, published by the vendor.
- [near-source] https://github.com/near/intents, observed by us on 2026-09-22.
- [sombra-verify-site] https://sombra.tech/, no date on page, published by the vendor.
- [sombra-verify-wp], [sombra-deposit-link], [sombra-crypto] https://sombra.tech/whitepaper.pdf, 2026-04-20, published by the vendor.
- [bonsol-groth16] https://docs.bonsol.org/core-concepts/introduction, no date on page, published by the vendor.
- [sombra-source] https://sombra.tech/, observed by us on 2026-09-22.
- [prior-iacr-2025-1741] https://eprint.iacr.org/2025/1741, 2025-09-24, measured by a third party.

We do not say Styx was the earliest to verify a STARK on Solana: a 2025 paper [prior-iacr-2025-1741] measured a STARK verification on Solana devnet at a mean of 1.10 × 10^6 compute units over 100 runs. [K100]

---

8. v2: the decisions and what each is designed to fix

v2 is **decided and not deployed**. Two of its work packages, WP2 and WP3 (the new hash; the new field and transcript), are written and tested, in a separate workspace that is not yet in the published repository: WP2, the Poseidon2 hash of the table below (width 12, 8 full and 22 partial rounds, the Horizen Labs constants, a four-element digest, the note, nullifier and depth-22 tree functions), and WP3, the cubic extension field and a new Fiat-Shamir transcript that draws its field challenges from it, with independent batching coefficients and 16-bit grinding. Their review gate of 2026-09-23 found no broken code (113 tests passed, 121 with every feature on; the hash agrees with Plonky3's Poseidon2 layers on 100,004 inputs and reproduces the Horizen Labs test vector) and still returned red, on one blocking item: a public hashing function of WP2 accepted domain tags reserved for nullifiers and tree nodes (not a collision, the gate notes, but a gap between the code and what its report claimed), plus missing tests. The two work packages then closed that item and the missing tests, each with a test that failed first (122 tests, 130 with every feature on). The gate's

checks were then re-run on the repaired code the same morning: the test suites (122 and 130 passed), the lint checks, the builds without the standard library and for WebAssembly, and the integration test (10 of 10) passed, and a deliberate reintroduction of the blocking item made the integration test fail, as it should; one build without the standard library, of the transcript with its SHA-256 dependency, stopped on a compiler error inside that dependency. No written verdict was recorded for that re-run. Nothing of v2 runs on chain, no v2 circuit exists yet, and each line below says what the design is meant to do. [K129] Every v2 figure is a calculator figure on **estimated** parameters (n = 1024 rows, 36 columns, 8 quotient segments, at most 64 constraints, taken from the design, not from code), or an estimate, and each is marked. The decisions are recorded in a decision record, and the founder confirmed its decisions D1 to D8, the eight rows of the table below, on 2026-09-23. [K101, K4]

DECISION	WHAT IT IS	WHAT IT IS DESIGNED TO FIX
Profile R	every challenge to be drawn uniformly from the cubic extension $F_p[x]/(x^3 - x - 1)$; rate 1/32; 36 queries; proof-of-work of 16 leading zeros (in plain words: challenges drawn from nearly 2^192 possible values instead of about 2^64, more redundancy in the encoded data, and 36 randomly chosen entries of the proof opened and checked instead of v1's 22 on most circuits)	the base-field challenges behind F52 and behind the low Johnson column of v1 [K23]
Poseidon2, width 12	a digest of four field elements (32 bytes) for commitments, nullifiers and tree nodes; 8 full and 22 partial rounds, the Horizen Labs instance, written and tested in WP2 (not deployed); a cryptanalysis memo of 2026-09-23 recommends keeping 22 partial rounds rather than 23, and names what would move it to 23 before the constants are frozen into the circuit	the one-element fingerprint behind F03, the double spend [K22, K129, K139]
Tree depth 22	2^22 notes per tree, the full path to be proven inside the circuit, so the pool program would do no hashing	room for 4,194,304 (2^22) notes per tree; it also sizes the circuit width (36 columns assume depth 22) [K102]
SHA-256 hash-chain token instead of C0	a single-use token bound to the subscriber for pausing and resuming a subscription	the C0 replay, in which anyone can pause or resume someone else's subscription with a copied proof (measured at 5,748 compute units for the pause) [K103]
New verifier program, pool upgraded in place	a new `p01_stark_verifier_v2` program; the pool gains v5 accounts; the v1 verifier stays frozen and reproducible	keeps the v1 verifier unchanged for the drain [K104]

DECISION	WHAT IT IS	WHAT IT IS DESIGNED TO FIX
One uniform shape for insert and spend	C6v2 and C7v2 to share the same size, parameters and padded compute; the compute padding relies on a Solana facility the design has not yet verified	the envelope that tells a deposit proof from a spend proof (ledger C1 to C4, B4), if the padding holds; the pool transaction would still show the operation type [K105]
90-day devnet drain of v1	v1 deposits frozen at the upgrade; v1 spends end after 90 days	ends v1; during the window F03, F52 and the C0 replay stay exploitable on v1, acceptable only because this is devnet; v1 never goes to mainnet [K106]
Inventory switch after the earliest accepted v2 spend	the inventory of issued notes (section 3.2; the design documents call it the "reserve") moves to v5 accounts, and issuance stops serving v1, once a v2 spend has been accepted on devnet	ties the switch to a checked fact, not a date; a v1 withdrawal followed by a v2 deposit is linkable by time, so a delay or a swap through the inventory is to be offered [K107]

The calculator's v2 figures (the shape shared by C6v2 and C7v2, uniform challenges, union bound, estimated parameters). [K108]

PROFILE	UNIQUE DECODING (THEOREM)	JOHNSON, BCIKS20 (THEOREM)	JOHNSON, BCHKS25 (THEOREM, 2025)	CONJECTURED, WITH THE SHA-256 CAP	QUANTUM: UD / JOHNSON BCIKS20 / CONJ.
Q (not chosen)	48.83	77.24	84.76	110.75	24.41 / 38.62 / 55.37
R (chosen)	50.39	**105.42**	105.88	173.64, capped at 128.00 by SHA-256	25.19 / 52.71 / 85.33
R-128 (fallback)	60.90	128.32, capped at 128.00	132.57, capped at 128.00	173.64, capped at 128.00	30.45 / 64.16 / 85.33

What this says and what it does not.

- The plan asks for at least 100 in the Johnson BCIKS20 (theorem) regime. On estimated parameters, R is calculated at 5.4 above it, and Q falls short. The figure to quote is "105.42, Johnson/BCIKS20, theorem, union bound, estimated v2 parameters", always with its quantum figure in the same regime next to it (52.71), never a bare number. [K108]
- The margin depends on the analysis parameter m = 56; the list size that m implies is not yet written down, and no test ties this column to the design. The calculator is to be re-run on the real circuit at the end of the circuit work; if the Johnson figure falls under 100, the query count rises toward 47 (R-128) rather than changing profile. The proof-format design of 2026-09-23 (not yet reviewed) also changes two of the calculator's inputs: how many functions are batched together, and the number of folds. The figures of this table have not been re-run with those inputs, and no v2 figure holds until they are. [K109, K149]
- Under the quantum line, no profile reaches 100: 52.71 for R in the Johnson regime. If the threat includes a quantum adversary against Fiat-Shamir, that is the figure to reason with. [K108]
- For a four-element digest, the calculator's generic collision line is 128.00 classical and 85.33 quantum, against 32.00 and 21.33 for v1's one-element fingerprint. That is a design figure for a hash not yet built, not a property of Styx. [K22]

Also required before any mainnet, from the audit: bind the insertion index in C6 (F01), bind the submitter or a key named by the proof in the spend and retire C1+C3 (F05, F55, F70), bind C0-style controls to a payer and a nonce (F25), canonicity checks on stored values (F28), an independent challenge per constraint (F02), the pool-program fixes (SPL fees taken in tokens rather than lamports, F30; prefunding of eras, F31; rebasing the root ring, F32; migration rent paid by the authority rather than taken from notes, F54), a redesign of `p01_liquidity` (F27), range checks before re-enabling C5 or deploying `p01_zkspl`, a corrected simulation argument (F69) and a published blinding bound (F66), then the external human review. [K110]

Not measured yet. Nothing of v2 runs on chain, and no v2 proof exists. Proof size, verification cost and latency are estimates. The figures below for size and verification (about 157,589 bytes in 42 chunks for R, with about 1.10 SOL of rent held while it is uploaded, and the 12-query shard below) are early estimates for the design document's proof layout, since superseded by a proof-format design (2026-09-23, in the same unpublished workspace) that has not been reviewed and that changes the proof layout, the number of chunks and the way verification is split; its own figures are design estimates too, and none of them is quoted here until it is reviewed. The base costs were measured, not on chain but in a local Solana VM (litesvm) running Solana bytecode compiled from the verifier's own arithmetic code: a Goldilocks multiplication at 76.93 compute units, a cubic-extension multiplication at 663.17, a SHA-256 Merkle node at 161. From them, one 12-query verification shard (the verification split into pieces that each fit one transaction) of the design document's layout is derived at about 0.99 million compute units, under the 1.4 million limit but more than the design's model assumed (an early estimate, superseded as above). With 16 leading zeros instead of 22, the expected proof-of-work falls from some 169 to 173 ms to some 2.6 to 2.7 ms natively,

derived from the measured cost of one attempt. Proving in 1 to 2 seconds is an estimate that depends on the transform-based prover already shipped. [K111, K112]

---

9. Reproducibility annex

The repository has been private since 2026-10-02 (section 9.5). [K118] The steps below record how each figure was produced, and someone with access to the repository can replay them.

9.1 Rebuild the prover byte for byte

The shipped prover `packages/stark-prover/wasm/p01_stark_bg.wasm` (240,172 bytes, SHA-256 `241caaabb505b44c0d58c06a603cdc4f9ac6b338b93405dabf2048e7ed3a41e9`) was rebuilt twice to identical bytes, with its glue and its three client copies. [K50, K113] To rebuild it:

A Windows x86-64 (MSVC) host. Cargo hashes the host triple into crate metadata; on an earlier blob (`0ad6d7f1`), a Linux rebuild came out one function shorter and 78 code bytes longer, and this was re-measured neither on `d5583d41` nor on `241caaab`, which were built with the same recipe. [K74]

rustc 1.98.1 with the `wasm32-unknown-unknown` target, wasm-pack 0.14.0, wasm-bindgen-cli 0.2.114 **built from crates.io** (`cargo install wasm-bindgen-cli --version =0.2.114 --locked`; the GitHub release binary stamps its commit and differs by 12 bytes), and binaryen version_117 ahead of any other binaryen on the PATH. [K113]

From the root of a checkout of the repository (private since 2026-10-02, section 9.5): `node scripts/ci/wasm-repro.mjs --runs 2`. It builds from clean target directories with one path remap (the blob embeds the builder's Cargo registry path in panic messages), and compares every rebuild with the shipped blob, its glue and the three copies the clients import (web, extension, mobile). Exit 0 means PASS. The same runs in CI as `.github/workflows/wasm-repro.yml` on a Windows runner. [K113, K74]

A proof from this blob was accepted on devnet (section 3.5); the offline record of that is checked by `node packages/stark-prover/scripts/deployed-verifier-check.mjs`, and `--verify-onchain` re-reads the deployment. [K55, K114]

9.2 Re-run the calculator

```sh
cargo run --manifest-path tools/security-levels/Cargo.toml -- --write   # regenerate
docs/SECURITY-LEVELS.md
cargo run --manifest-path tools/security-levels/Cargo.toml -- --terms   # every error term,
round by round
cargo run --manifest-path tools/security-levels/Cargo.toml -- --prose   # every figure quoted
in the docs, and its verdict
cargo test --locked --manifest-path tools/security-levels/Cargo.toml    # fails if the
document or the prose drifts
```

[K61, K115]

9.3 Re-run the tests

The CI workflow `.github/workflows/ci.yml` is the reference list. Its main steps: `cargo test --release -p p01-stark` (the prover), `cargo test -p p01_stark_verifier --lib` and the soundness pins, `cargo test -p zk_shielded` suites, the security-levels tests above, `pnpm turbo run test` (the TypeScript packages and the web app), and `node verify/p01-verify.mjs --self-test --replay verify/fixtures/v4-live` (the on-chain linkage probe against recorded devnet data, with a positive control). The on-chain programs are built for Solana and exercised in litesvm by `.github/workflows/sbf-litesvm.yml`. [K116]

9.4 Re-run the benchmark

From a clean checkout at the measured commit, after `pnpm install`:

```sh
npx tsx scripts/bench/run.mts --only local --node-accepted <major, if not Node 24>
npx tsx scripts/bench/run.mts --only all --dry-run --key <dedicated devnet keypair> \
  --cluster devnet --rpc-env P01_BENCH_RPC --api-base <deployment URL> --balance <SOL>
npx tsx scripts/bench/run.mts --only all --key <dedicated devnet keypair> \
  --cluster devnet --rpc-env P01_BENCH_RPC --api-base <deployment URL> \
  --rpc-plan '<provider and plan tier>' --node-accepted <major, if not Node 24>
```

The dry run sends nothing and prints the SOL the run needs (about 125 devnet SOL for the proof pipeline and the four product flows at n = 30). Use a key made for the benchmark only, never one linked to anything else. [K117]

9.5 Access to the code and licence terms

Since 2026-10-02 the repository is private and the code is not published. Copies published before that date shipped with the licence file of their commit: the MIT License for commits and `@protocol-01` npm versions published before 2026-09-22 (`LICENSE-MIT-BEFORE-POLYFORM`), and the PolyForm Strict License 1.0.0, copyright Volta Team, from the commit that replaced the MIT License (`LICENSE`). [K118] The benchmark takes everything that differs from one person to the next (key, RPC, deployment) as arguments. [K119]

Evidence for every fact above: `docs/CLAIMS.md`. Security figures: `docs/SECURITY-LEVELS.md`. Benchmark method: `docs/BENCHMARK-METHOD.md`. Leak ledger: `docs/LEAK-LEDGER.md`.