FIX VALIDATION REPORT / 07 SEPTEMBER 2026

A boundary-safe range-proof cache key for Elements 23.x

FIX TARGET

Exact `elements-23.x` commit `323fc71625755917a54a5759875939805eee53c1`. One-file cache-key repair; no release or recovery claim.

THE INVARIANT

A cache hit must identify one complete verification statement

CURRENT 323fc716

```
proof || value || asset || script
```

FRAME FIELDS →

PROPOSED

```
[len][proof] ... [len][script]
```

Independent forensic review by Alpen Labs Agentic Security; not an official Blockstream, Liquid Federation, or SideSwap report.

SECTION 01

Executive Summary

> *Recommendation: apply the reviewed* `LE64` *field-framing diff to the exact* `elements-23.x` *target, retain the exact incident regression, and treat canonical-chain recovery as a separate change.*

The target branch hashes four range-proof verification inputs as adjacent raw bytes: the proof, value commitment, effective asset-generator bytes, and output script. Because the proof and script are variable-length, two different verification statements can have the same cache preimage. A cache hit then returns success before parsing and cryptographic verification. This is an encoding ambiguity, not a SHA-256 collision or a failure of the range-proof cryptography.

The proposed one-file change prefixes every field with an eight-byte little-endian length. That makes field boundaries part of the cache identity. The exact patch applies cleanly to commit `323fc716...` and produces byte-for-byte the source that was compiled and tested.

The exact branch control is decisive:

Question	Result	Evidence
Does unmodified `323fc716...` reproduce the bad cache decision?	Yes	3/3 fresh runs accepted the hot target; each exited `201` with 10/11 assertions
Does the proposed framing block the transfer?	Yes	5/5 fresh patched runs rejected the hot target; each passed 11/11 assertions
Does the patched branch pass its broad C++ unit invocation?	Yes, within recorded coverage	All 480 executed cases passed; the recorded final run passed 8,085,603 assertions

Validation is strongest at the exact branch and transaction path

Each row answers a different question; unexecuted release gates remain explicit.

01	SOURCE IDENTITY	Exact 323fc716 raw source	byte-for-byte match
02	RED / GREEN	Raw accepts; patch rejects	3/3 RED | 5/5 GREEN
03	FULL INVOCATION	All 480 executed cases pass	8,085,603 assertions*

NOT VALIDATED HERE

- production binaries
- full ConnectBlock / IBD
- functional test suite
- fuzz + sanitizers
- multi-architecture builds
- historical recovery rule

* Recorded final run; assertion totals vary because the suite contains randomized tests.

One fixture-data test was skipped because DIR_UNIT_TEST_DATA was unset.

Figure 1. The branch result is strong for the exact source and transaction path. Production deployment, full historical replay, fuzzing, sanitizers, and multi-architecture packaging were not validated here.

This report is deliberately branch-specific. It does not establish which binary ran in production, a wider affected-version interval, a fixed release, or a canonical-chain recovery rule.

SECTION 02

Background

The cache invariant

Elements verifies a confidential output's range proof against four inputs:

- range-proof bytes;
- value-commitment bytes;
- effective asset-generator bytes, passed through the `asset_commitment` argument; and
- the output script used as committed extra data.

The range-proof cache is a performance optimization. A successful check with `store=true` may populate the cache. Later validation can reuse that result. The security invariant is simple:

> *A cached success may be reused only for the same complete four-field verification statement.*

The per-process random salt prevents precomputation across processes. It cannot distinguish two tuples that already produce the same field-concatenation bytes before the salt and range-proof domain prefix are applied.

The exact target

At 323fc71625755917a54a5759875939805eee53c1`, `src/script/sigcache.cpp:75-77, `CSignatureCache::ComputeEntryRangeProof` contains this construction:

```
CSHA256 hasher = m_salted_hasher_range_proof;
hasher.Write(proof.data(), proof.size())
      .Write(commitment.data(), commitment.size())
      .Write(asset_commitment.data(), asset_commitment.size())
      .Write(scriptPubKey.data(), scriptPubKey.size())
      .Finalize(entry.begin());
```

The four `Write` calls feed one hash stream. They do not encode field tags or boundaries.

Cache lookup and lazy discard marking

`CachingRangeProofChecker::VerifyRangeProof` computes the entry and checks the cache before it parses the fields or invokes `secp256k1_rangeproof_verify`:

```
if (rangeProofCache.Get(entry, !store)) {
    return true;
}
```

For `store=false`, `Get` calls `contains(entry, true)`. In this CuckooCache implementation, that marks a hit as eligible for later reclamation; it does not immediately delete or hide the entry. Without an intervening insertion, the same key remains findable. The exact RED/GREEN test preserves this behavior.

SECTION 03

Vulnerability Details

One byte stream, two different statements

The retained incident tuples have these field lengths:

Statement	Proof	Value commitment	Effective asset-generator bytes	Script	Raw total
Valid primer	4,166	33	33	69	4,301
Invalid target	4,234	33	33	1	4,301

The complete 4,301-byte field concatenations are byte-for-byte equal even though their semantic boundaries differ. SHA-256 of the retained unsalted concatenation is `82b0b8ccf8c743171f2bc8d80bb9982a81cfca583e8dfbe65563cf095e99c01a`. This comparison value is not the process-local cache-entry hash, which also includes a random nonce and domain prefix. Prepending the same process prefix preserves equality.

The repair makes field boundaries part of the identity

The incident resegmented one byte stream. LE64 framing makes the two tuples encode differently.

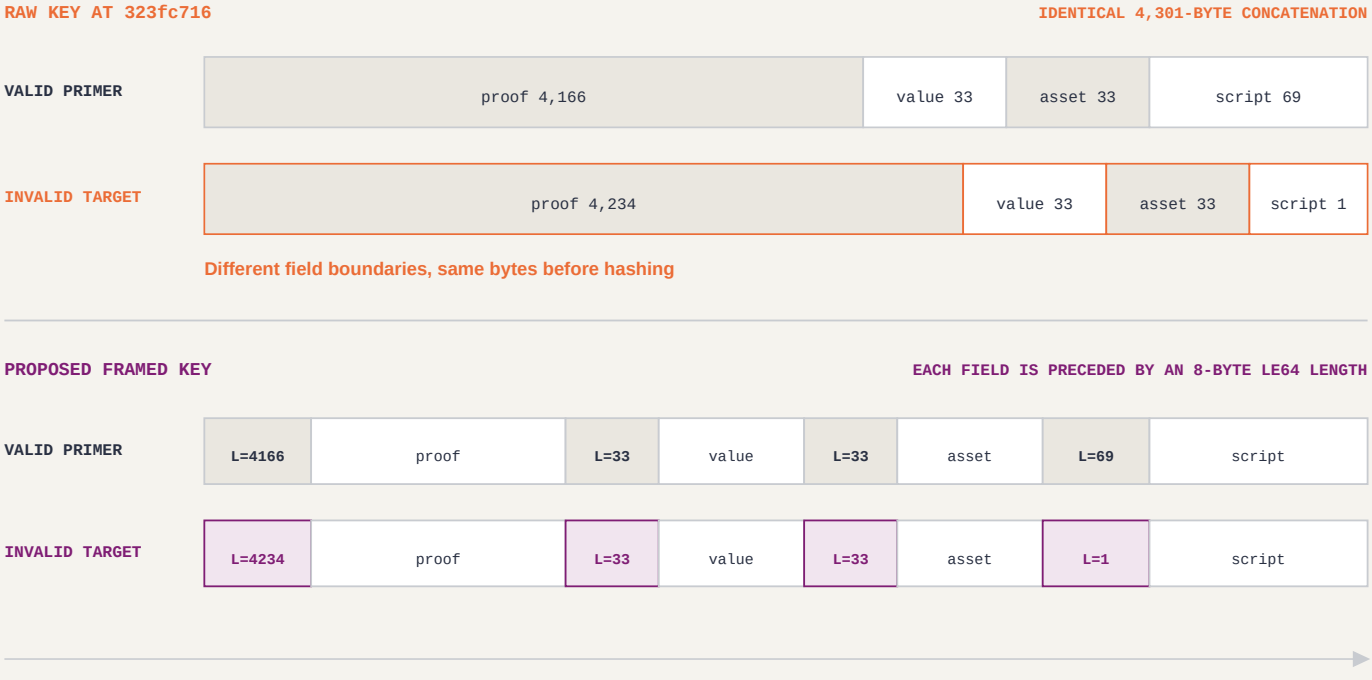


Figure 2. The current key hashes an untyped byte stream. The repair writes an LE64 *length before every field, so moving bytes across a boundary changes the preimage before hashing.*

Why the early return matters

The valid primer passes fresh verification and populates the relevant matching cache entry. A second transaction with the same valid statement reuses that entry; it is not an independent verification or insertion. Under the raw encoding, the distinct invalid target maps to the same entry and receives cached success. A fresh check rejects the target.

The exact `elements-23.x` raw control reproduced that distinction: the cold target rejected, while the hot target was accepted after priming. The proposed framing gives the target a different identity, causing the verifier to run and reject it.

SECTION 04

Exploitability Analysis

Demonstrated conditions

The exact branch replay establishes the following narrow primitive:

1. A valid statement is checked with cache storage enabled.
2. A different invalid statement has the same unframed four-field byte concatenation.
3. The target reaches the same process while the matching cache entry remains findable.
4. The lookup returns success before fresh range-proof verification.

The retained transaction sequence satisfies those conditions in the controlled branch test. Cache eviction or unrelated insertions can disrupt a real-world sequence; the replay does not measure reliability under production traffic.

Evidence classification

Classification	What this review supports
Confirmed	Raw key at exact `323fc716...`; retained tuple-concatenation equality; exact raw RED; exact patched GREEN; recorded source, object, test-source, binary, and diff hashes
Proven by construction	`LE64` framing makes the four-field encoding unambiguous: equal framed preimages require equal lengths and equal field bytes
Source-backed inference	When the amount and script checks execute, the strict `ConnectBlock` path propagates the repaired `CheckTxInputs` rejection
Unknown	Production binary and deployment inventory; historical cache traces; cache retention under real network traffic
Not validated	Full `ConnectBlock`, initial block download, reindex recovery, functional tests, fuzzing, sanitizers, and multi-architecture packaging

Historical-chain consequence

The retained invalid target is in Liquid block 4,050,336. The repaired `Consensus::CheckTxInputs` replay rejects the transaction with `bad-txns-in-ne-out, value in != value out`. Source inspection shows the normal block-connection path propagates that failure when the relevant checks execute, but this review did not run a complete historical `ConnectBlock` replay.

Replacing the process with a patched binary starts with an empty process-local cache. It does not, by itself, define how a strict historical replay should treat an already-recorded block.

SECTION 05

Proof of Concept

Exact RED/GREEN method

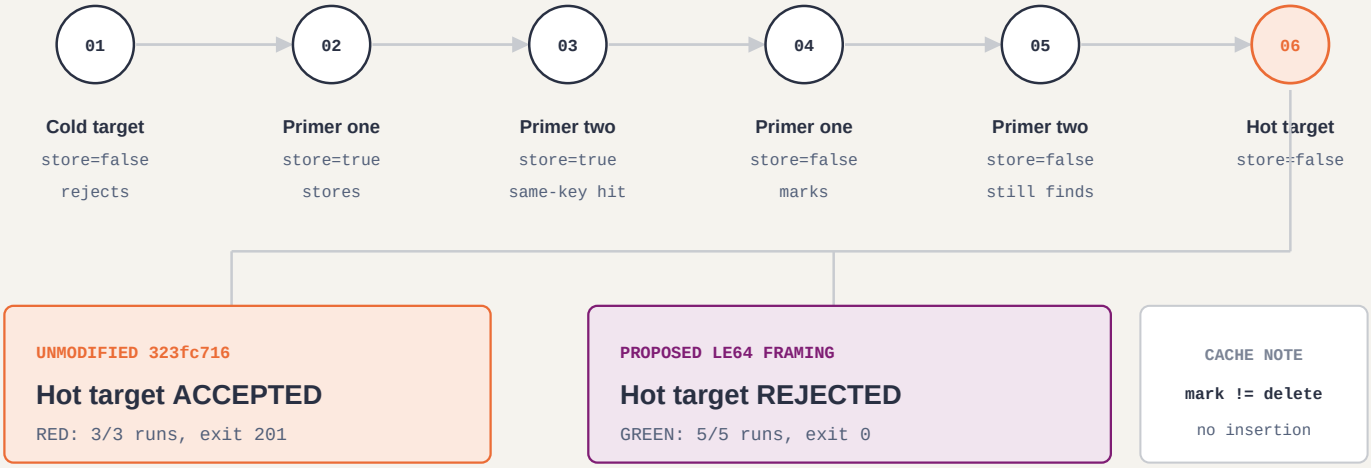
No standalone transaction-construction program is included. The validation-only Boost test loads retained public transaction bytes and calls the production `Consensus::CheckTxInputs` path with the retained parent outputs available. The same test source is compiled into raw and patched binaries.

The sequence is identical in both binaries:

1. Cold target with `cache_store=false`.
2. Primer one with `cache_store=true`.
3. Primer two with `cache_store=true`.
4. Primer one with `cache_store=false`.
5. Primer two with `cache_store=false`.
6. Hot target with `cache_store=false`.

The exact test changes only the cache-key construction

Both binaries process the same retained transaction sequence with block-connection cache flags.



The final target uses store=false, matching the block-connection lookup mode.

Figure 3. The final target uses the block-connection lookup flag. Lazy discard marking leaves the colliding raw entry findable because no insertion occurs between marking and the target lookup.

Observed results

Build	Fresh runs	Cold target	Hot target	Assertions	Exit
Exact unmodified `323fc716...`	3	Rejected	Accepted	10/11	201
Proposed `LE64` framing	5	Rejected	Rejected	11/11	0

Both patched target rejections were `bad-txns-in-ne-out, value in != value out`. The empty-field case also returned `false` without a crash; source inspection confirms that the `field.empty()` guard skips the data write.

Broad branch invocation

After the final relink, a full `test_bitcoin` invocation passed all 480 executed cases. The recorded final run passed 8,085,603 executed assertions and exited zero. Assertion totals vary because the suite includes randomized tests. `script_tests/script_assets_test` was skipped because `DIR_UNIT_TEST_DATA` was unset.

The test fixture is supplied outside the Elements tree. A portable invocation uses a local evidence-root placeholder rather than an author-machine path:

```
env LIQUID_REVIEW_ROOT=<liquid-review-root> \
  ./src/test/test_bitcoin \
  --run_test=incident_length_prefix_fullpath_tests
```

Build and runtime validation used one host/toolchain: Apple clang 14.0.0, arm64, macOS 26.6.2. Multi-architecture and package validation remain open release gates.

SECTION 06

Remediation

Exact proposed implementation

The product-source diff is limited to `src/script/sigcache.cpp`. It adds the project `WriteLE64` helper and replaces the raw write chain with this implementation:

```
CSHA256 hasher = m_salted_hasher_range_proof;
static_assert(sizeof(size_t) <= sizeof(uint64_t));
// Frame each byte string so field boundaries are part of the cache key.
const auto write_field = [&hasher](const auto& field) {
    unsigned char length[8];
    WriteLE64(length, static_cast<uint64_t>(field.size()));
    hasher.Write(length, sizeof(length));
    if (!field.empty()) {
        hasher.Write(field.data(), field.size());
    }
};
write_field(proof);
write_field(commitment);
write_field(asset_commitment);
write_field(scriptPubKey);
hasher.Finalize(entry.begin());
```

The resulting suffix is:

```
LE64(len(proof)) || proof
|| LE64(len(commitment)) || commitment
|| LE64(len(asset_commitment)) || asset_commitment
|| LE64(len(scriptPubKey)) || scriptPubKey
```

Equal framed encodings require equal field lengths and equal field bytes. The static assertion prevents length truncation on supported platforms, and the empty-field guard avoids an empty-container data write.

Patch identity

Artifact	SHA-256
Unmodified `sigcache.cpp`	`4616c187411e265d568d5c393a0a3ef2a31990bcb2fb607a2c05810c82d22a62`
Patched `sigcache.cpp`	`56389c3d18c40279a5a41fdcd9e4a95164ac211ea83eafd18c21aed832b4cf10`
Recommended diff	`05554a25818b2036bb4f8beff0e3b68b52cd24e2b8931aab3070dc96a07d8edc`
Validation test source	`6348700e0ccbb33a961a587915c36ca5d8925d1dc85610d0b29736f16ccb77bc`
Patched test binary	`02810adbed8f97768846da67dc5495e90272641e4cbba80dedd7a73406d8f406`

The recommended diff applies cleanly to exact commit `323fc716...` and yields a `sigcache.cpp` byte-identical to the tested patched source.

Deployment is not recovery

Figure 4. The validated source change addresses future cache identity. Canonical history still requires its own reviewed and tested policy.

For forward safety, operators need coordinated upgraded processes and a restart so no process retains entries derived from the old encoding. The cache is process-local and does not require database migration.

Historical recovery is a separate decision. Candidate policy families include a coordinated reorg that removes the invalid transition or a narrowly reviewed Liquid-mainnet historical exception bound to exact identities. This report neither exhausts nor endorses those choices, and the proposed diff implements neither.

Remaining release gates

- retain the exact incident regression in a portable fixture;
- add same-tuple equality and single-field inequality unit tests;
- run fixture-data tests with `DIR_UNIT_TEST_DATA` available;
- complete functional, sanitizer, fuzzing, packaging, and supported-architecture validation;
- validate coordinated restart and deployment procedures; and
- choose and independently validate the historical-recovery rule before claiming full network recovery.

SECTION 07

Summary

The exact `elements-23.x` target at commit `323fc716...` uses an ambiguous range-proof cache-key encoding. The retained incident provides distinct valid and invalid statements with the same raw field concatenation. The exact unmodified branch accepts the invalid target after priming; the proposed `LE64` framing causes fresh verification and rejection.

The candidate is ready for branch review: it is a one-file change, applies cleanly, compiles on the recorded host, passes the exact RED/GREEN control, and passes all 480 cases executed by the recorded full `test_bitcoin` invocation. The evidence does not yet support a release, deployment, multi-architecture, or full historical-recovery claim.

> *Decision: use the provided diff as the* `elements-23.x` *fix candidate, preserve the exact branch regression, and keep deployment approval separate from canonical-history recovery approval.*

Evidence record

- Branch validation receipt: `evidence/patch-validation-2026-09-07/elements-23.x/validation_receipt.json`
- Recommended branch diff: `evidence/patch-validation-2026-09-07/elements-23.x/recommended-fix.diff`
- Retained collision receipt: `evidence/chain/ce4cae-ancestry/c26d719-cache-collision.json`
- Retained raw-key replay: `evidence/chain/ce4cae-ancestry/actual-c26d719-cache-replay.json`

Independent forensic review by Alpen Labs Agentic Security; not an official Blockstream, Liquid Federation, or SideSwap report.