Verifiable Anonymous Voting on Top of One-Time Traceable Ring Signatures: Two Architectures and a Comparison

Research Artifact, v0.4

Daniele Lin Niccolò Pagano

May 2026

Abstract

We present a verifiable anonymous voting artifact built on Scafuro and Zhang's one-time traceable ring signature scheme (OTRS) [18] and instantiated over Ristretto255 with RFC 9380 hash-to-curve. On top of that single cryptographic primitive we implement and compare *two* system-layer architectures. **Architecture A** is a hash-chained append-only bulletin board whose entries are co-signed by a t-of-N Ed25519 publisher cohort and checkpointed by a k-of-M witness federation that surfaces cohort equivocation publicly — a federated, Certificate-Transparency-style system. **Architecture B** is a Bitcoin-shaped proof-of-work chain with permissionless mining, an eVote ledger, typed voting transactions, and the same OTRS-based ballot semantics. Both architectures support the same election lifecycle (Setup → Registration* → Ring → Ballot* → Closed → Tally?) and reuse the same record schema. The artifact ships 127 tests — including 16 covering the PoW path end-to-end — two CLIs, and a documented threat model. We compare the architectures on trust model, decentralisation, cost, censorship resistance, and cross-poll linkability, and argue that the two answer different questions: A trades permissionless validator membership for cheap, auditable accountability against a small named authority set; B trades that for open mining at the cost of bringing back the security-budget asymmetry that makes blockchain voting fragile [20]. Residual gaps — coercion resistance, true threshold-Ed25519 (FROST), formal verification, post-quantum variants — are flagged for future work.

1 Introduction

Verifiable anonymous voting has become a recurring research theme because nation-scale deployments still rely on trust-the-server architectures whose properties are not publicly checkable [17]. The cryptographic literature has long offered alternatives—Helios [1], Civitas [4], end-to-end verifiable schemes such as Scantegrity [3] and, more recently, ElectionGuard [2]—but each makes architectural commitments that constrain the kinds of elections they fit.

Ring signatures, introduced by Rivest, Shamir and Tauman [16], offer a particularly appealing primitive for ballot anonymity: a signer proves membership in a ring of public keys without revealing which member they are. The challenge for voting is that a *plain* ring signature lets a voter sign repeatedly. Two parallel literatures address this: *linkable* ring signatures [11, 8] which expose a tag identifying re-use of the same key, and *traceable* ring signatures [6, 18] which expose the actual public key on misuse.

This artifact implements Scafuro and Zhang's 2021 scheme [18]. We chose it because (i) its trace algorithm is deterministic and one-time-tag-based, not interactive; (ii) it requires no trusted setup; (iii) the algebra is a clean Σ-OR proof over a single prime-order group, which makes it a realistic target for both implementation and future formal verification.

1.1 Research question

Once the cryptography is in place — and the cryptography is the expensive intellectual capital — the natural follow-on question is *system layer*: who stores ballots, who orders them, who is trusted to do so, and what evidence does an auditor need to refute a dishonest publisher? In particular: does *replacing a federated log with a proof-of-work chain buy you anything that meaningfully changes the security or operational story for OTRS-based voting?* This is the question we set out to answer empirically.

We build both architectures on top of the same OTRS primitive and the same record schema, then compare them on five axes drawn from the e-voting literature: trust model, decentralisation, censorship resistance, finality, and cross-poll linkability. We further evaluate end-to-end cost (wall-clock and storage) at matched voter counts. section 7 states the answer up front — the chain buys permissionless validator membership at a real but bounded operational cost, and *nothing* for properties that live in the OTRS layer. The detail follows from section 5 through section 10.

1.2 Contributions

1. **OTRS reference implementation** (`otrs/`, ≈1 100 LOC Python) of KeyGen, Sign, Verify, Traceover Ristretto255, with all randomness drawn from `os.urandom` and all hashing routed through a domain-separated RFC 9380 hash-to-curve.

2. **Architecture A — federated bulletin board** (`voting/`, ≈1 800 LOC): a publisher-signed, hash-chained, append-only log; typed records for the election lifecycle (Setup → Registration* → Ring → Ballot* → Closed → Tally); a t-of-N publisher cohort with asynchronous threshold co-signing; a k-of-M witness federation that checkpoints log heads and makes cohort equivocation publicly detectable; manager, voter, witness, and auditor principal APIs; and a CLI for the full flow.

3. **Architecture B — proof-of-work chain** (`chain/`, ≈1 400 LOC): a Bitcoin-shaped SHA-256 PoW chain with heaviest-chain selection and median-time difficulty adjustment; an account ledger with nonces, fees, and a coinbase reward; eight typed transactions (coinbase, transfer, plus (setup_poll, register_voter, publish_ring, ballot, close_poll, tally) that wrap the same record schema as Architecture A); a permissionless miner; a chain-aware auditor; and a CLI parallel to A's.

4. **Comparison** of the two architectures on trust model, decentralisation, cost, censorship resistance, and cross-poll linkability (section 7).

5. **Threat model** (`paper/threat_model.md`) covering both architectures.

6. **Test suite** (`tests/`, 127 tests): algebraic invariants of the group wrapper, hash-to-curve determinism and domain separation, OTRS sign/verify correctness, negative cases, property-based tests under Hypothesis, bulletin-board integrity, threshold cohort and witness behaviour, end-to-end Architecture A election flow, adversarial cases (double-sign, forged attestation, tampered ballot, wrong publisher key, extra records, ballot outside the ring, illegal choice), and end-to-end Architecture B blockchain flow (PoW, ledger arithmetic, nonce replay, tx signature tampering, on-chain double-sign detection, save/load round-trip).

7. **Microbenchmark** (`bench/`) for the cryptographic primitive (section 10).

8. **Critique and revival** of the prior implementation in `legacy/` — a 160-bit DSA-style subgroup with `random.randrange` randomness, no tests, and a half-built PoW ledger. Architecture B is in part the legacy ambition modernised: same PoW backbone, hardened crypto, and the missing pieces (eVote ledger, typed voting transactions, audit pipeline) filled in.

1.3 Non-goals

We do not claim a side-channel-hardened implementation: Python's `int` arithmetic is not constant-time, and our wrapper around libsodium has a small number of branches on potentially derived scalars (always with negligible collision probability). We also do not deliver a formal proof in EasyCrypt or ProVerif; both are flagged as next steps (section 12).

2 Preliminaries

We work in a prime-order cyclic group $\mathbb{G}$ of prime order q with generator g, in which the Decisional Diffie-Hellman (DDH) problem is assumed hard. We instantiate $\mathbb{G}$ as Ristretto255 [5], a prime-order group of order

$$q = 2^{252} + 27742317777372353535851937790883648493$$

built on top of Curve25519. Ristretto eliminates the small-subgroup and cofactor pitfalls of raw Edwards25519, ruling out a class of implementation bugs by construction.

We require three hash functions modelled as random oracles:

$$H_0 : \{0,1\}^* \to \mathbb{G}, \quad H_1 : \{0,1\}^* \to \mathbb{G}, \quad H_2 : \{0,1\}^* \to \mathbb{Z}_q.$$

Both H_0, H_1 are instantiated as the RFC 9380 [15] random-oracle hash-to-curve for Ristretto255 with the suite identifier `ristretto255_XMD:SHA-512_R255MAP_RO_`. H_2 uses the same XMD expander followed by `crypto_core_ristretto255_scalar_reduce` (uniform on $\mathbb{Z}_q$ within statistical distance 2^{-126}).

Per RFC 9380 best practice we tag each oracle with a distinct domain-separation string fixed in `otrs/otrs.py`:

```
DST_H0 = b"OTRS-v1-RISTRETTO255-XMD:SHA-512-H0"
DST_H1 = b"OTRS-v1-RISTRETTO255-XMD:SHA-512-H1"
DST_H2 = b"OTRS-v1-RISTRETTO255-XMD:SHA-512-H2"
```

Any change to the algebraic structure (curve, hash, transcript format) *must* bump the version prefix.

Assumption 1 (DDH in $\mathbb{G}$)**.** For uniformly random $a, b, c \in \mathbb{Z}_q$, the distributions (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are computationally indistinguishable.

3 The Scafuro–Zhang construction

Fix an *issue* I—in the voting application, the election identifier—and a ring $R = (pk_1, \ldots, pk_n)$ of n public keys. We use 1-indexed positions so we can take modular inverses; $i \in \{1, \ldots, n\}$ is invertible in $\mathbb{Z}_q$ since $n \ll q$.

KeyGen. $x \xleftarrow{\$} \mathbb{Z}_q^*$, $pk = g^x$.

Sign(x_i, I, m, R, i)**.**

1. Compute $h \leftarrow H_0(I \,\|\, R)$ and $A_0 \leftarrow H_1(I \,\|\, R \,\|\, m)$.
2. Compute the trace tag $\sigma_i \leftarrow h^{x_i}$ (this is what makes the scheme one-time-traceable: σ_i depends only on (I, R, x_i), not on m).
3. Set $A_1 \leftarrow (\sigma_i / A_0)^{1/i}$. For $j \neq i$, define $\sigma_j \leftarrow A_0 \cdot A_1^j$; observe σ_i also satisfies the recurrence by construction.

4. For $j \neq i$ sample $c_j, z_j \xleftarrow{\$} \mathbb{Z}_q$ and set $a_j \leftarrow g^{z_j} \cdot pk_j^{c_j}$, $b_j \leftarrow h^{z_j} \cdot \sigma_j^{c_j}$.

5. For $j = i$ sample $w \xleftarrow{\$} \mathbb{Z}_q$ and set $a_i \leftarrow g^w$, $b_i \leftarrow h^w$.

6. Compute
$$c^* \leftarrow H_2(I \,\|\, R \,\|\, A_0 \,\|\, A_1 \,\|\, a_1 \,\|\, \cdots \,\|\, a_n \,\|\, b_1 \,\|\, \cdots \,\|\, b_n).$$

7. Set $c_i \leftarrow c^* - \sum_{j \neq i} c_j$ and $z_i \leftarrow w - c_i x_i$.

8. Output $\sigma \leftarrow (A_1, c_1, \ldots, c_n, z_1, \ldots, z_n)$.

Verify(σ, I, m, R)**.** Recompute h, A_0, $\sigma_j = A_0 \cdot A_1^j$, $a_j = g^{z_j} \cdot pk_j^{c_j}$, $b_j = h^{z_j} \cdot \sigma_j^{c_j}$ for all j, and the challenge c^* as above. Accept iff $\sum_j c_j \equiv c^* \pmod q$.

Trace$(R, I, (m_1, \sigma^{(1)}), (m_2, \sigma^{(2)}))$**.** Recompute h, then $A_0^{(k)} = H_1(I \,\|\, R \,\|\, m_k)$ for $k \in \{1, 2\}$, and $\sigma_j^{(k)} = A_0^{(k)} \cdot (A_1^{(k)})^j$ for $j = 1, \ldots, n$. For the *real* signer at position i^* we have $\sigma_{i^*}^{(k)} = h^{x_{i^*}}$ for both k (the trace tag is m_k-independent), so column i^* matches. For $j \neq i^*$ the tag depends on m_k, so columns generically differ. Outcome:

- **exactly one** column matches ⇒ that column is the double-signer;
- **all** columns match ⇒ the two signatures are identical (replay or signature on the same message twice—we label this "linked");
- **no** columns match ⇒ independent signers.

This is implemented in `otrs.otrs.trace`.

4 Implementation

4.1 Layout

```
otrs/
  group.py        # Ristretto255 wrapper: Scalar, Point, base_mul,
    ↪ ORDER
  hash.py         # RFC 9380 XMD:SHA-512 expand + hash-to-curve,
    ↪ hash-to-scalar
  otrs.py         # keygen / sign / verify / trace
  serialize.py    # canonical ring + signature encodings
  _libsodium.py   # direct cffi binding to libsodium.so.23
tests/            # unit, property, serialization tests
bench/            # microbenchmarks
paper/            # this document
legacy/           # the prior implementation, preserved for reference
```

4.2 Group wrapper

`otrs.group` exposes Ristretto255 through `Scalar` and `Point` frozen dataclasses with byte-level canonical encodings (32 bytes each). All operations delegate to libsodium via a direct cffi ABI binding (`otrs/_libsodium.py`); we did not rely on PyNaCl because the Ubuntu-shipped build does not export Ristretto255 bindings. The `Point` constructor calls `crypto_core_ristretto255_is_valid_point` so caller-supplied encodings cannot bypass group-membership checks.

4.3 Hash-to-curve

We implement `expand_message_xmd` (RFC 9380 §5.3.1) over SHA-512 in pure Python (≈25 lines) and compose it with `crypto_core_ristretto255_from_hash` to obtain the random-oracle hash-to-curve. The same expander, followed by `crypto_core_ristretto255_scalar_reduce`, gives a uniform-by-statistical-distance hash-to-scalar. The DST-oversize path (`H2C-OVERSIZE-DST-` prefix, RFC 9380 §5.3.3) is implemented and tested.

4.4 Randomness

All secret randomness flows from `os.urandom` via `Scalar.random()`, which pulls 64 bytes and reduces mod q. This is the only RNG path in the artifact—there is no rejection-sampled custom PRG, no `random.randrange`, no seedable fallback. The legacy implementation, by contrast, sampled keys with `random.randrange` (predictable from a Mersenne-Twister state) and implemented a hash-based PRG that is incompatible with the random-oracle assumption.

4.5 Canonical encoding

The transcript hashed into H_2 depends bit-for-bit on the encoding of the ring and the auxiliary points (a_j, b_j). We commit to the encoding in `otrs/serialize.py`: a 4-byte length prefix, then concatenated 32-byte Ristretto encodings. Order is significant, so permuting the ring changes the signature; this is required for the OR-proof structure.

5 System architecture

The cryptographic primitive in section 3 gives us *ballot anonymity* and *double-vote detection*. Turning that into a usable election system requires three additional pieces: a tamper-evident public record of everything that happened, a typed schema for what can validly appear on that record, and well-defined roles for the principals who interact with it. `voting/` delivers these.

5.1 Bulletin board

The bulletin board is an *append-only, hash-chained, cohort-signed* log. Each entry has the shape

$$\mathsf{Entry} = (i, h_{prev}, t, payload, \{(k_1, \sigma_1), \ldots, (k_s, \sigma_s)\})$$

where i is a contiguous index starting at 0, h_{prev} is the SHA-256 hash of entry $i-1$ (or all-zero for the genesis entry), t is a Unix timestamp, *payload* is opaque bytes interpreted by the record schema below, and the final set is a collection of *cohort-member signatures*: each σ_j is an Ed25519 signature by cohort member k_j over the canonical content preimage. An entry is valid only when at least t distinct cohort members have signed it, where t is the threshold pinned in the genesis :class:‘ElectionSetup‘. The entry hash is computed over the content alone, so collecting additional signatures does not perturb the chain.

This is the Certificate-Transparency / Sigsum / C2SP transparency-log model rather than a proof-of-work blockchain. Elections do not face Sybil-mining attacks; they face misbehaving administrators. A threshold-signed log is simpler, audited, and fits the actual threat model. The cohort layer gives *liveness against* $N - t$ *unavailable publishers* and *soundness against* $t - 1$ *corrupted publishers*. Equivocation *within* the cohort ($\geq t$ corrupted) is addressed by witnesses (section 5.3).

5.2 Record schema and state machine

Each payload is a canonical JSON object discriminated by a `kind` field. Six record kinds form a state machine:

$$\mathsf{Setup} \to (\mathsf{Registration})^* \to \mathsf{Ring} \to (\mathsf{Ballot})^* \to \mathsf{Closed} \to \mathsf{Tally}?$$

Setup pins the election parameters: a 32-byte random `election_id` that doubles as the OTRS *issue* value, the manager's Ed25519 public key, ballot options, and the schedule (`registration_close` $\leq$ `voting_open` $\leq$ `voting_close`). Registration contains a voter's Ristretto255 OTRS public key, a human-readable handle, and an Ed25519 *attestation* by the manager over `voter-attestation-v1` $\|$ `election_id` $\|$ pk $\|$ `handle`. Ring is the canonical-order list of attested voter pks; after it is published, registration is closed and voting opens. Ballot carries an OTRS signature over `ballot-v1` $\|$ `choice`, bound to the issue and the ring. Closed is a marker. Tally is the manager's claimed result — which any auditor can independently recompute.

5.3 Witness federation

The cohort signs entries; *witnesses* sign *heads*. A witness is an independent third party with their own Ed25519 keypair. Periodically (in the artifact, on-demand via the CLI) each witness fetches the bulletin board, verifies it under the cohort, and emits a *checkpoint*

$$\mathsf{Checkpoint} = (\mathit{log\_index},\ \mathit{head\_hash},\ \mathit{witness\_index},\ \sigma_{\mathsf{Ed}})$$

signed under the witness's key with the domain-separated preimage `otrs-witness-checkpoint-v1 || log_index || head_hash || witness_index`. Checkpoints accumulate in a sidecar file `<log>.witnesses` that travels alongside the log.

Witnesses give us equivocation resistance: if a corrupted cohort ($\geq t$ malicious members) publishes two divergent logs, honest witnesses who see both will sign two different `head_hash` values for the same `log_index`. The auditor surfaces this as hard *equivocation evidence* — a publicly verifiable proof that the cohort split. The genesis ElectionSetup pins the witness set and a k-of-M threshold; the auditor requires at least one log index to be co-signed by $\geq k$ distinct witnesses (all agreeing on the hash) before declaring the election trustworthy.

5.4 Principals

Publisher cohort (`voting/manager.py`). The N cohort members jointly publish Setup, register voters, freeze the Ring, accept ballots, close voting, and publish a Tally. Any subset of size $\geq t$ may sign each entry, either synchronously (single-process tests) or asynchronously through the pending-entry sidecar (`<log>.pending`) for the multi-process / multi-machine deployment. The cohort is *not* trusted to compute the tally honestly: any tally they publish is a claim that auditors recompute and refute.

Voter (`voting/voter.py`). Generates a Ristretto255 OTRS keypair, sends the public key to the cohort out-of-band for inclusion, fetches the bulletin board, and, once the ring is published, signs a ballot for their chosen option using OTRS Sign and submits it.

Witnesses (`voting/witness.py`). Each holds an Ed25519 key, fetches and verifies the log, and emits a checkpoint co-signing the current head. Witnesses are *read-only* — they cannot append to the main log — and their security guarantee is that as long as k of them are honest, equivocation by the cohort is publicly detectable.

Auditor (`voting/auditor.py`). Anyone with the bulletin board can run the auditor; the cohort and witness identities are bootstrapped from the genesis Setup entry itself, so the only out-of-band trust is the channel by which an auditor obtained that entry (typically a printed-on-a-poster cohort identity or a well-known URL). The auditor:

1. verifies the chain (indices contiguous, prev-hashes match, $\geq t$ valid cohort signatures per entry, timestamps non-decreasing);
2. enforces the state machine, rejecting out-of-order records;
3. requires the published ring to equal the multiset of voter pks appearing in cohort-attested Registration entries;
4. OTRS-verifies every ballot against the issue, message, and ring;
5. if witnesses are declared, verifies their checkpoints (signatures valid, no equivocation, $\geq k$ distinct witnesses co-sign at least one log index);
6. groups ballots by signer using the OTRS trace columns (section 3), counts a single vote per voter, and excludes ballots from any voter who signed two distinct messages;
7. compares the canonical tally against the cohort's Tally record, if present.

The CLI exposes the auditor as `evote audit` and exits non-zero on any failure or tally mismatch.

6 Architecture B: proof-of-work chain

`chain/` implements the same election lifecycle on top of a Bitcoin-shaped proof-of-work chain. The cryptographic primitive (section 3) and the record schema (`voting/records.py`, section 5.2) are reused *verbatim*. What changes is the storage and ordering layer: instead of a t-of-N cohort co-signing every entry, anyone may mine, and the canonical history is the chain with the greatest cumulative $2^{\text{difficulty}}$ (GHOST-style heaviest-chain selection).

6.1 Block and chain format

A block has a 121-byte fixed header

$$(prev\_hash,\ height,\ difficulty,\ timestamp,\ miner\_pk,\ tx\_root,\ nonce)$$

plus a variable-length transaction list. The PoW puzzle is the standard SHA-256-of-header inequality:

$$\mathsf{leading\_zero\_bits}(\mathrm{SHA}{-}256(header)) \geq difficulty.$$

Transactions are bound into the header via $tx\_root = \mathrm{SHA}{-}256(\|_k \mathrm{canon}(tx_k))$ (a flat hash rather than a Merkle root — block bodies are small enough that inclusion proofs are unnecessary in the prototype). Difficulty is adjusted ± 1 when the median inter-block interval over a sliding window of 11 blocks falls outside $[0.5T,\ 1.5T]$ around a target $T = 60$ seconds, mirroring the legacy node's adjustment rule.

6.2 Account ledger and transactions

Every Ed25519 public key maps to an account with a *balance* and a *nonce*. Eight transaction kinds are defined; six wrap the `voting/records.py` record types so the per-record validation already proved correct in section 5.2 is reused:

Kind	Signed by	Wraps
`coinbase`	none (per-block, first tx)	— (mints reward + fees)
`transfer`	sender	— (eVote payment)
`setup_poll`	poll creator	ElectionSetup
`register_voter`	sponsor	VoterRegistration
`publish_ring`	poll creator	RingPublication
`ballot`	*none*	Ballot
`close_poll`	poll creator	VotingClosed
`tally`	poll creator	TallyPublication

The `ballot` kind is the only transaction with *no* chain-layer sender, signature, or fee. This is essential for anonymity: a fee-bearing ballot would expose the voter's Ed25519 identity on chain. Validity is the OTRS signature inside the payload, verifiable by anyone against the published ring. Outside the ring, no valid signature can be produced, so flooding the mempool with bogus ballots is computationally bounded by ring membership and rejected at validation; miners are expected to include all valid ballots they see.

6.3 State machine

The same lifecycle as Architecture A is enforced at the chain layer. `chain.state` maintains a per-poll `PollState` indexed by `election_id`, with phase transitions registration → voting → closed → tallied triggered by the corresponding transaction kinds. Per-tx checks include: poll-creator identity match (only the address that issued the `setup_poll` may issue ring / close / tally), per-poll registration uniqueness, ring membership equals the multiset of attested registrations, ballot timestamp within [voting_open, voting_close], and OTRS verification of every ballot at the moment it is included in a block.

Block application is atomic: a snapshot of the state is taken at entry, mutations happen on the snapshot, and the snapshot is committed only on success — so a malformed transaction does not leave the chain in a partial state.

6.4 Mining and node lifecycle

The miner (`chain/mining.py`) takes a header template (with `nonce` = 0) and scans nonces until the PoW threshold is met. For the artifact prototype this is single-process; a production miner would partition the nonce space across cores or machines.

`chain.node.Node` owns the canonical chain and the folded state; `append(block)` validates and applies; `save/load` persist the chain as a length-prefixed binary stream and rebuild state from scratch on load (so a corrupted state cache cannot silently desync from chain content). Fork resolution by cumulative weight is *not* implemented in this prototype — the node accepts the chain it is given; the necessary information (`State.cum_weight`) is available and a heaviest-chain switch is a small extension.

6.5 Audit

`chain.auditor.audit_chain` loads the chain through `Node.load`, which already enforces PoW, signatures, nonces, balances, the state machine, and OTRS-per-ballot validity. The auditor then runs the same σ-column bucketing as `voting.auditor.audit` (section 5.1) to compute the canonical tally, identify double-signers, and compare against any on-chain TallyPublication. The tally algorithm is byte-identical between the two architectures because it depends only on the ballots and the ring, not on how either of them got there.

6.6 CLI walk-through

Each subcommand that produces a new state mines its transaction into a fresh block on top of the current tip — one CLI call equals one block. This is simpler than a mempool for demo purposes and makes the demonstration auditable line-by-line.

```
python3 -m chain.cli account-keygen --out alice.json
python3 -m chain.cli voter-keygen   --out v0.json
python3 -m chain.cli init-chain --chain c.bin --miner miner.json \
    --allocate "$(jq -r .pk_b64 alice.json):1000" --timestamp $T0
EID=$(python3 -m chain.cli setup-poll --chain c.bin --miner miner.json
   ↪ \
    --creator alice.json --title "Demo" --options "yes,no" \
    --registration-close $T1 --voting-open $T2 --voting-close $T3)
python3 -m chain.cli register-voter --chain c.bin --miner miner.json \
    --sponsor alice.json --election-id $EID \
    --voter-pk $(jq -r .pk_b64 v0.json) --handle v0
python3 -m chain.cli publish-ring --chain c.bin --miner miner.json \
    --creator alice.json --election-id $EID
python3 -m chain.cli vote --chain c.bin --miner miner.json \
    --voter v0.json --election-id $EID --choice yes
python3 -m chain.cli close-poll --chain c.bin --miner miner.json \
    --creator alice.json --election-id $EID
python3 -m chain.cli audit --chain c.bin
```

The audit prints the chain head, the per-poll tally, and any double-sign culprits, and exits non-zero on any inconsistency.

7 Architecture comparison

The two architectures share an entire cryptographic stack — the same OTRS primitive, the same record schema, the same canonical encodings, the same audit-side algorithm for tally and trace. What differs is the storage and ordering layer, and that single delta cascades into the operational properties below.

7.1 When each architecture wins

Architecture A wins when the election has a known, accountable authority set (a university senate, a corporate board, an inter-institution treaty body), when ballots need to be finalised in seconds rather than minutes, and when the operational footprint must not exceed the named cohort plus a handful of independent witnesses. It is essentially the Certificate-Transparency / Sigsum / sigstore pattern applied to elections: small named log + threshold trust + independent witnesses for equivocation defence. No tokens, no fees, no mining, no chain reorganisation.

Architecture B wins when permissionless validator membership is a non-negotiable design constraint (e.g. no party — including the poll creator — is trusted to host the canonical record), when many polls are expected to share the same infrastructure and pay for its operation, and when the operator is comfortable accepting probabilistic finality and a non-trivial cost-per-vote. The chain has no genesis-pinned trust set beyond *the rules of the protocol*.

7.2 When neither wins

Both architectures expose persistent identities at the chain layer: in A the cohort and witness public keys, in B the poll creator and sponsor public keys (and any Ed25519 addresses used to fund eVotes). Neither architecture solves *coercion resistance* — a voter who reveals their OTRS secret to a coercer enables trace-tag matching in either system. Neither defends against

Table 1: Architecture A (federated bulletin board) vs. B (PoW chain).

Property	**A: federated** (`voting/`)	**B: PoW chain** (`chain/`)
Validator membership	permissioned: N named publishers at genesis	permissionless: anyone with a miner
Liveness	against $N-t$ unavailable publishers	against any minority hashpower
Safety from forks	threshold-signed total order; no forks	probabilistic, heaviest chain wins
Censorship resistance	any t honest publishers can publish	any miner can include any valid tx
Equivocation defence	witness federation, k-of-M checkpoints	longest-chain rule; reorgs always possible
Sybil resistance for voters	cohort-signed registration	poll-creator-signed registration (same)
Cost per ballot	1 SHA-256 + threshold Ed25519 sigs	1 block of PoW + tx validation
Wall-clock to close	seconds	minutes (target block interval)
Disenfranchisement risk	zero (voters do not transact)	non-zero (sponsor or self pays fees)
Cross-poll linkability	per-election ring; no global identity	persistent Ed25519 addresses for poll creators / sponsors; voter OTRS keys can be rotated per poll
Receipt-freeness	open (same in both)	open (same in both)
Required infrastructure	$N + M$ named parties + their key custody	miners + an eVote bootstrap distribution
Failure mode if abandoned	log goes stale, audit still works	chain stops; abandoned blocks rewritable by anyone with hashpower
Lines of code	≈1 800 + 111 tests	≈1 400 + 16 tests

compromised voter devices — a malicious client trivially signs ballots the user did not authorise. Both gaps are independent of the storage layer; they sit upstream of any OTRS-based voting design and require JCJ/Civitas-style coercion-resistance machinery to close.

7.3 The economic asymmetry, addressed honestly

In A the security budget is the operational cost of running N publishers plus M witnesses — a one-off setup expense, bounded. In B the security budget per block is the fee-plus-reward paid to miners, which over a window is bounded by aggregate poll fees on the chain. Crucially, the *attacker's* payoff scales with the largest poll on the chain at any moment, not with average fees. For elections of meaningful political stakes (corporate boards, public referenda) a sufficiently funded adversary can rent hashpower for the voting window at a cost below their expected gain from controlling outcome [20]. Architecture A sidesteps this entirely because it does not try to be Sybil-resistant via economic incentive — it relies on *naming* the trust set and making cheating by that set publicly evident. Whether that trade is acceptable is a deployment-specific judgment, not a cryptographic one.

8 Threat model (summary)

The full threat model lives in `paper/threat_model.md`. Briefly, under DDH in Ristretto255, the random-oracle model for SHA-512–based hash-to-curve, and Ed25519 unforgeability, v0.3 provides: *(i)* anonymity inside the ring for any single ballot; *(ii)* public, deterministic detection of double-voting within an election; *(iii)* publicly recomputable tallying; *(iv)* tamper-evident history; *(v)* eligibility enforcement against the cohort's attestation set; *(vi)* liveness against $N - t$ unavailable cohort members; *(vii)* soundness against $t - 1$ corrupted cohort members; and, when

witnesses are configured, *(viii)* publicly verifiable equivocation evidence whenever the cohort itself splits. It does *not* provide receipt-freeness (a voter who reveals x_i to a coercer enables trace-tag matching), defence against compromised voter devices, or post-quantum security. These remaining gaps are the v0.4 roadmap.

9 Cryptographic security

The scheme is proved unforgeable, anonymous, and traceable in the random oracle model under DDH [18]. We do not reproduce the proofs here; we restate the security claims as the implementation enforces them and flag where our instantiation deviates from the paper's abstract model.

- **Unforgeability** follows from soundness of the Σ-OR proof: producing a valid $(\{c_j\}, \{z_j\})$ tuple with $\sum_j c_j \equiv H_2(\cdot)$ without knowing any x_j requires inverting H_2 or solving discrete logarithm.
- **Anonymity** follows from honest-verifier zero-knowledge of each individual Σ-proof, plus DDH for the trace tag σ_i.
- **Traceability** follows because $\sigma_{i^*} = h^{x_{i^*}}$ is fully determined by (I, R, x_{i^*}) and independent of m.

Our instantiation matches the abstract assumptions modulo two choices: the hash-to-curve is *modelled* as a random oracle (XMD:SHA-512 RO is the cryptographic-community-standard instantiation), and the group is prime-order Ristretto255 rather than an abstract prime-order group (a strictly safer instantiation than the original DSA subgroup, ruling out small-subgroup and cofactor attacks by construction).

9.1 Known limitations

- **Constant-time.** Python integer arithmetic is not constant-time; the point–scalar product delegates to libsodium which *is* constant-time. The remaining timing channel is the `is_zero` branch in `Point.scalar_mul`, which is reached only with probability $\approx 2^{-252}$ for random scalars derived in our protocol—negligible.
- **Equality of group elements** uses Python `bytes` equality, which is not constant-time. Group-element equality is *not* secret-dependent in our verification path (both sides are derived from public inputs), so this is acceptable.
- **No HSM-style binding** between a signer's secret and the host: a compromised client trivially signs anything.

Remark 1 ($n = 1$ is degenerate)**.** The trace algorithm's "all columns match" and "exactly one column matches" predicates coincide for a ring of size 1. In that case Trace returns "linked" even on a double-sign attempt. Anonymity is undefined for $n = 1$ regardless, so callers should require $n \geq 2$ for any realistic use.

10 Evaluation

Microbenchmarks were collected on a single core (no parallelism) on a Linux laptop, with five repetitions per operation, median reported.

10.1 Cryptographic primitive

Reproduce with:

```
python3 -m bench.bench_otrs --sizes 2,4,8,16,32,64,128 --output
    ↪ bench/results.csv
```

Table 2: OTRS cost vs. ring size n.

n	Sign (ms)	Verify (ms)	Trace (ms)	Sig size (B)
2	0.67	0.72	0.42	164
4	1.77	1.72	0.68	292
8	2.62	3.14	1.32	548
16	5.20	5.79	2.57	1 060
32	10.42	10.51	5.27	2 084
64	20.77	20.89	10.77	4 132
128	45.02	42.70	20.37	8 228

Cost is linear in n—empirically about 0.34 ms of sign time and 0.33 ms of verify time per ring member at this configuration, with the constant dominated by the $2n + 1$ scalar multiplications each side performs. Trace runs in ≈ 0.16 ms per member because it computes only n scalar multiplications per signature rather than $2n + 1$ exponentiations plus group additions. Signature size is $|A_1| + 4 + 2n \cdot |\text{scalar}| = 32 + 4 + 64n$ bytes, matching the measured column to the byte.

10.2 Full election

The end-to-end cost of an election is dominated by the OTRS work: each ballot needs one Sign call (signer side, $O(n)$) and one Verify call (auditor side, $O(n)$). The bulletin-board layer adds only a SHA-256 hash, an Ed25519 signature, and a file append per record, all in the microsecond range. table 3 shows the wall-clock cost of a complete election for varying voter counts N, where every registered voter casts one ballot and the auditor verifies the entire log.

Table 3: Full election cost (single core, ring size = number of voters).

N	Register (ms)	Ring publish (ms)	All ballots (ms)	Audit (ms)	Log size (KB)
10	6.4	0.9	41.3	51.2	21.2
25	15.1	0.9	245.6	277.7	93.0
50	45.1	1.5	1 004.7	1 104.0	323.8

Ballot submission and audit are both $O(N^2)$ in this configuration because the ring grows with N and the per-ballot cost is $O(\text{ring size})$. For larger elections one would either (i) shard voters into multiple parallel rings, or (ii) move to a logarithmic-size traceable ring (section 12, item 1). At $N \leq 50$ the entire end-to-end flow finishes in under 1.2 seconds on a single core; the bulletin-board log fits in a third of a megabyte.

10.3 Full election — Architecture B (chain)

For Architecture B we re-run the same election at the same voter counts, with one CLI invocation per transaction and PoW pinned to 8 bits (256 expected SHA-256 hashes per block — sufficient for benchmark purposes, with the difficulty-cost relationship measured separately in table 5). table 4 reports the end-to-end wall-clock; PoW dominates the per-block fixed cost, while the OTRS-bound per-ballot cost remains the same as in A.

Table 4: Architecture B end-to-end cost at PoW difficulty $d = 8$, single-core single-process miner. Reproduce with `python3 -m bench.bench_chain -voters 5,10,25,50 -difficulty 8`.

N	Register (ms)	Ring publish (ms)	All ballots (ms)	Audit (ms)	Chain size (KB)
5	6.2	1.4	21.5	13.8	11.6
10	10.9	1.3	88.7	47.4	26.3
25	27.2	1.4	469.5	282.6	104.0
50	47.5	1.0	1 798.0	1 114.1	344.4

Table 5: PoW cost vs. difficulty bits at single-core SHA-256 throughput, median over 5 trials. Reproduce with `python3 -m bench.bench_chain -pow-only -difficulty 4,8,12,16`.

Difficulty bits d	Expected hashes 2^d	Mine time / block (ms)
4	16	0.09
8	256	0.95
12	4 096	12.57
16	65 536	264.57

What the comparison shows. At matched N, Architecture B costs roughly **1.6**× more end-to-end than A (at $N = 50$, the chain finishes the same election in ~ 3 seconds versus 1.1 seconds for the federated log). The gap is dominated by two effects: one Coinbase + tx-list root hash per block adds a small fixed cost, and ballots travel through full transaction validation (signature parse, OTRS verify, state-machine dispatch) rather than the lighter cohort-signed log append. The per-ballot OTRS work itself is identical between architectures because the primitive is shared.

Storage scales similarly: at $N = 50$ the chain weighs 344 KB vs. 324 KB for the federated log — a $\sim 6\%$ overhead, attributable to per-block headers and coinbase transactions. Audit cost is roughly comparable (1.1 s on chain vs. 1.1 s on log); on the chain it is dominated by re-running every block's full validation including PoW recomputation.

PoW extrapolation. table 5 shows the single-core Python miner sustains $\approx$ 270,000 SHA-256 hashes per second. A native miner (Rust/C, optimised) on the same hardware would hit $\sim$ 50M hashes per second — roughly 200,000× faster — so production deployment at Bitcoin-style difficulty ($\sim$ 80 bits today) is no more practical for an OTRS-voting chain than it is for any small chain. The Bitcoin-style attack-economics calculus in section 7 therefore applies independently of implementation effort: rented hashpower wins because the chain's fee-funded security budget cannot keep up with the political stakes of even a single significant poll.

Reading the comparison. The end-to-end numbers do not by themselves recommend one architecture over the other: a 1.6× slowdown is comfortably acceptable at the voter counts where either system makes sense ($N \leq 100$). The decisive comparison is in section 7: A buys you cheap auditable accountability against a small named set; B buys you permissionless validator membership at the cost of bringing back the standard PoW-voting attack surface [20].

11 Comparison to related work

A spectrum of ring-signature constructions exists, parametrised by signature size, signer/verifier cost, trust assumptions, and traceability properties.

The relevant axis for our use case (one ballot per voter, public auditability, no trusted authority) is *traceability*, which is rarer than linkability. Most linkable schemes only reveal

Table 6: Ring-signature design space.

Scheme	Sig size	Trace	Setup	Assumption
RST01 [16]	$O(n)$	none	none	RSA-like
LSAG/CLSAG [8]	$O(n)$	linkable	none	DDH
MLSAG [12]	$O(n)$	linkable	none	DDH
Triptych [13]	$O(\log n)$	linkable	none	DDH
Lelantus [9]	$O(\log n)$	linkable	none	DDH
Raptor / Falafl [14]	$O(\log n)$	linkable	none	Lattice (PQ)
Scafuro–Zhang [18]	$O(n)$	**traceable (one-time)**	none	DDH, ROM

that two signatures share a signer without identifying them; OTRS additionally identifies the misbehaving public key. The cost is signature size: $O(n)$ versus $O(\log n)$ for state-of-the-art linkable schemes.

A natural research direction (section 12) is to combine the trace tag with a log-size set-membership proof à la Groth–Kohlweiss [7] or Triptych.

12 Open problems and future work

1. **Logarithmic-size traceable rings.** Replace the linear Σ-OR with a Groth–Kohlweiss-style membership proof [7] while keeping the trace tag $\sigma_i = h^{x_i}$. The challenge is binding the per-position A_1 reconstruction to the index that the membership proof commits to.

2. **Formal verification.**

   - *EasyCrypt*: model the three games (unforgeability, anonymity, traceability) and machine-check the reductions to DDH and ROM. The scheme's small algebraic surface makes this tractable.
   - *ProVerif/Tamarin*: model the *protocol layer* of the e-voting application—registration, ballot submission, tracing—and verify ballot anonymity and double-vote detection as observational equivalence properties.

3. **Post-quantum variants.** The DDH reduction breaks under Shor's algorithm. Lattice-based linkable rings (Raptor, Calamari, Falafl) exist; constructing a *traceable* lattice ring with one-time tags is, to our knowledge, open. A literature survey plus barrier analysis is a tractable Master's thesis topic.

4. **Coercion resistance.** OTRS prevents *vote duplication* but not *vote selling*—a coerced voter could prove to a coercer which signature is theirs by revealing x_i. A receipt-free extension à la JCJ [10] or Civitas [4] is needed for coercion resistance. This is the single most important gap in v0.2; see `paper/threat_model.md` §4.

5. **True threshold Ed25519 (FROST).** v0.3's t-of-N threshold is implemented as a vector of t separate Ed25519 signatures, one per signer. This is the convention used by CT and Sigsum and is sound, but a single FROST [19] aggregated signature would compress entries from $64t$ to 64 bytes. A clean follow-up is to swap the vector for FROST without changing the rest of the system.

6. **Gossip protocol for bulletin board distribution.** v0.3 treats the log and the witness sidecar as local files. A production deployment would replicate them across the cohort and a wider set of mirrors, with auditors fetching from ≥ 2 independent mirrors and cross-checking. This is engineering work rather than research, and is independent of the cryptography.

7. **Cross-validation against RFC 9380 test vectors** for ristretto255 hash-to-curve; we ship regression vectors but not (yet) the official ones, since RFC 9380's appendix omits ristretto255 vectors in some normative drafts.

13 Reproducibility

This artifact is released under the MIT license at https://github.com/NickP005/e-ring-voting.

```
# system deps (Debian/Ubuntu)
sudo apt install -y libsodium23 python3-cffi python3-pytest \
    python3-hypothesis python3-cryptography

# full test suite: 127 tests across both architectures + crypto +
   ↪ properties
python3 -m pytest -v

# OTRS benchmarks
python3 -m bench.bench_otrs --sizes 2,4,8,16,32,64,128

# Architecture A: federated bulletin-board demo
python3 -m voting.cli manager-keygen --sk-out msk.pem --pk-out mpk.pem
python3 -m voting.cli setup --log log.jsonl --sk msk.pem --title
   ↪ "Demo" \
    --options "yes,no,abstain" \
    --registration-close <T0> --voting-open <T1> --voting-close <T2>
# ...voter-keygen / register / publish-ring / vote / close / tally...
python3 -m voting.cli audit --log log.jsonl --pk mpk.pem

# Architecture B: PoW chain demo
python3 -m chain.cli account-keygen --out alice.json
python3 -m chain.cli voter-keygen   --out v0.json
python3 -m chain.cli init-chain --chain c.bin --miner alice.json \
    --allocate "$(jq -r .pk_b64 alice.json):1000" --timestamp <T0>
# ...setup-poll / register-voter / publish-ring / vote / close-poll...
python3 -m chain.cli audit --chain c.bin
```

References

[1] B. Adida. Helios: Web-based Open-Audit Voting. USENIX Security, 2008.

[2] J. Benaloh et al. ElectionGuard: a Cryptographic Toolkit to Enable Verifiable Elections. 2021.

[3] D. Chaum et al. Scantegrity II. EVT, 2008.

[4] M. Clarkson, S. Chong, A. Myers. Civitas: Toward a Secure Voting System. IEEE S&P, 2008.

[5] H. de Valence et al. The Ristretto Group. https://ristretto.group.

[6] E. Fujisaki, K. Suzuki. Traceable Ring Signature. PKC, 2007.

[7] J. Groth, M. Kohlweiss. One-Out-of-Many Proofs. EUROCRYPT, 2015.

[8] B. Goodell et al. Concise Linkable Ring Signatures and Forgery Against Adversarial Keys. 2020.

[9] A. Jivanyan. Lelantus: Toward Confidentiality and Anonymity of Blockchain Transactions. 2019.

[10] A. Juels, D. Catalano, M. Jakobsson. Coercion-resistant electronic elections. WPES, 2005.

[11] J. K. Liu, V. K. Wei, D. S. Wong. Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups. ACISP, 2004.

[12] S. Noether. Ring Signature Confidential Transactions for Monero. 2015.

[13] S. Noether et al. Triptych: Logarithmic-Sized Linkable Ring Signatures. 2020.

[14] X. Lu, M. H. Au, Z. Zhang. Raptor: A Practical Lattice-Based (Linkable) Ring Signature. ACNS, 2019.

[15] S. Scott, S. Wood, R. Wahby, A. Faz-Hernández, N. Sullivan. Hashing to Elliptic Curves. RFC 9380, 2023.

[16] R. Rivest, A. Shamir, Y. Tauman. How to Leak a Secret. ASIACRYPT, 2001.

[17] D. Springall et al. Security Analysis of the Estonian Internet Voting System. CCS, 2014.

[18] A. Scafuro, B. Zhang. One-time Traceable Ring Signatures. ESORICS, 2021.

[19] C. Komlo, I. Goldberg. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. SAC, 2020.

[20] S. Park, M. Specter et al. Going from bad to worse: from Internet voting to blockchain voting. Journal of Cybersecurity, 2021.