

On 4/9/2025

ME:

Insiders2TrustPoster_3_9_25.pdf

Proving_vote_correctness_in_the_IVXV_internet_voti.pdf

benchmarkverkle_2504.14069v1.pdf

the poster in the first file suggest using Verkle Trees for proving the number and integrity of votes in the Estonian i-voting system IVXV; the paper on the second file although solve a different problem (invalid votes) has a general opinion of avoiding them "While tree-based approaches can yield constant-size proofs with fast verification times, they generally require the use of general-purpose zk-SNARKS³². As such, they do not satisfy our selection criteria. We also did not pursue pairing-based ZKSMs due to the additional hardness assumptions of pairings and since the ZKRP approach seemed promising". On the other hand, the benchmarks in the third file (although talks about blockchains) shows that Verkle Trees give a fast construction time of nearly 1 second as long as $n < 2^{22}$ (for IVXV the average number of votes is less than 1 million). So, given the 3 files how do judge the applicability of Verkle Trees on IVXV? is there some complications I've missed about the elliptic curves used or any other cryptographic detail?

Copilot:

Applicability of Verkle trees to IVXV

Direct answer: yes, Verkle trees are applicable to IVXV for integrity- and count-style consistency checks, but only with careful choices around the commitment scheme, the "what exactly is committed" model, revoting semantics, and deployment optics. The strongest design for IVXV avoids general-purpose zk-SNARKs and also avoids pairing-based KZG unless you are willing to accept a trusted setup and new hardness assumptions. If you stick to DLOG-only assumptions, use an IPA/Pedersen-style Verkle construction over a well-vetted curve. Oberst's 1-second-scale proving for $n \leq 2^{22}$ maps well to IVXV's scale, so performance is not your bottleneck; the real complexity lies in modeling "count" and "consistency over time" without leaking linkability or overhauling IVXV's crypto stack.

Key cryptographic choices

- **Commitment scheme (the crux):**
 - **Option A — KZG-based Verkle (pairings):**

- Pros: tiny proofs, very fast verification, widely engineered in Ethereum contexts.
- Cons: introduces pairings, a structured (trusted) setup, and a new assumption stack. This conflicts with the “DLOG-only” preference used in your vote-correctness paper and adds governance risk around toxic waste.
- **Option B — IPA/Pedersen-based Verkle (no pairings):**
 - Pros: no trusted setup, sticks to DLOG, matches the philosophy behind Bulletproofs-based work.
 - Cons: larger proofs and somewhat slower verification than KZG, but still easily within IVXV’s performance envelope.
- Recommendation: if the political/assurance posture is “no new hardness assumptions and no trusted setup,” go IPA/Pedersen. If you can justify a robust, multi-party, auditable trusted setup and the benefit of small proofs outweighs the optics, KZG is viable.
- **Curve selection:**
 - **For KZG:** BLS12-381 (pairing-friendly) is the de facto choice; this diverges from IVXV’s P-384 universe and from Estonian ID’s signatures. That’s acceptable as a self-contained module, but it’s a separate crypto domain to audit, deploy, and maintain.
 - **For IPA/Pedersen:** choose a mainstream, audited curve with mature libraries (e.g., secp256k1, Ristretto255/Curve25519, or P-256/P-384). If you want 128-bit soundness without awkward challenge sizing, P-384 is conservative and politically familiar in Estonia; Ristretto255 is fast and developer-friendly but will force some parameter compromises if you need 256-bit challenges.
 - Recommendation: if avoiding pairings, pick a curve with a strong industrial footprint and a healthy library ecosystem you can certify and freeze early.
- **What Verkle actually guarantees (subtle but important):**
 - **Inclusion/consistency:** You get strong authentication of the key–value map across versions and short witnesses for membership/updates.

- **“Number of votes” is not automatic:** A vector commitment does not inherently prove cardinality. You must design for it:
 - Fix a public index space with a defined “empty” value and commit to a sparse map; then separately prove the count (e.g., a committed occupancy bitmap with an auditable sum, or maintain a committed counter that’s updated alongside inserts/deletes with consistency proofs).
 - Or publish an append-only log with epoch roots and prove monotonic growth plus a reconciliation rule for deletions (revotes) that preserves “counted-as-collected.”
 - Recommendation: model “count” as a first-class committed object (e.g., a committed counter + consistency proofs) rather than relying on heuristics like “open many leaves and infer cardinality.”
-

Fit with IVXV’s workflow and revoting

- **Revoting semantics:**
 - **Labeling moves:** Your poster’s VT_1/VT_2/VT_3 decomposition is sound in spirit but must be encoded precisely:
 - **Final tally set (VT_final):** the set of i-votes that survive paper overrides and re-votes; publish roots per processing step and at close of voting.
 - **Superseded set (VT_deleted):** per-voter “multiple” votes with proofs of deletion linking from the superseded vote to the final one; enables sampled audits without deanonymizing.
 - **Invariant proofs:** prove $n(\text{VT_cast}) = n(\text{VT_final}) + n(\text{VT_deleted})$ and that each deletion is tied to a valid supersession chain.
 - **Privacy:**
 - Never commit to raw identifiers; commit to salted, domain-separated hashes of ballot identifiers managed by the processor, and separate time granularity to mitigate correlation attacks from TMS timing.

- If QR codes expose “number of multiple votes,” ensure they are detached from identity-bearing artifacts; otherwise they risk coercion leakage.
 - **Bindings to existing crypto:**
 - **No decryption, no mixing changes:** Verkle runs on commitments/ hashes of encrypted-ballot records; it does not touch your ElGamal or mix-net.
 - **Vote-correctness proofs coexist:** Your Bulletproofs-based correctness proofs live in clients/collector; the Verkle layer authenticates the collector’s evolving ballot set. They are orthogonal and compatible.
 - **Public verifiability scope:**
 - Publish periodic Verkle roots and epoch witnesses to let any auditor verify append-only growth and post-close invariants.
 - For deeper audits, provide sampled membership/opening proofs under NDA that don’t reveal identities (e.g., audit the occupancy bitmap sum; check random supersession links in VT_deleted).
-

Performance and operations

- **Scale match:** With $< 1\text{M}$ i-votes per election, you’re well below the regimes where Verkle proof/witness handling becomes a bottleneck. The “order of seconds” witness construction and sub-MB proof payloads in benchmarking contexts translate favorably for IVXV’s cadence.
- **Online operation:** Keep the Verkle structure in memory during voting with durable snapshots of roots and update logs, or back it by a fast KV store with a write-ahead log. You don’t need Ethereum-grade engineering; an election-scope tree is modest.
- **Concurrency and failure modes:**
 - **Atomic updates:** Treat “collector accepts vote” as a transaction that updates both the ballot store and the Verkle state, emitting an update proof (or deferrable batch proof) with a crash-consistent WAL.
 - **Batching:** If single-vote witnesses feel heavy, commit per time-slice (e.g., every second) and publish the new root + batched multiproof.

- **DoS:** Cap per-epoch proof generation and rate-limit adversarial burst submissions; Verkle verification remains lightweight.
-

Risks and gotchas to address up front

- **Assumption mismatch:** If you use KZG, you introduce pairings and a trusted setup, which your other IVXV work explicitly avoided. If that contrast becomes a talking point, it can undermine the “removing insiders’ trust” message unless you run an exemplary, externally monitored ceremony.
 - **“Counting” fallacy:** Don’t overclaim that a Verkle proves “the number of elements” unless you explicitly commit to an occupancy structure or a counter and prove consistency. Without that, Verkle proves authenticated membership and evolution, not cardinality by itself.
 - **Timing service correlation:** Aligning TMS timestamps across government logs and IVXV supports consistency checks but risks deanonymization vectors if not coarsened and randomized. Bucket times and apply k-anonymity thresholds for any public correlation claims.
 - **Deletion semantics and leakage:** Supersession proofs must not enable linking final ballots to earlier submissions in a way that survives mix-net anonymization; keep all links at the level of committed identifiers only visible to the processor/auditor, with public proofs about counts and structure, not identities.
 - **Library maturity and audits:** Whichever Verkle stack you choose, freeze versions early, commission third-party audits, and avoid “research-grade” variants (e.g., exotic curves or bespoke multiproof gadgets) in a voting system.
-

Recommended path for IVXV

- **Define the committed state clearly:**
 - **Labels:** Commit to records of the form $C = H(\text{domain} || \text{election_id} || \text{salt_epoch} || \text{vote_record_id} || \text{payload_hash})$.
 - **Trees:**
 - VT_cast (all accepted i-votes), VT_final (eligible i-votes after overrides), VT_deleted (superseded i-votes).

- A committed occupancy bitmap or a committed counter with per-update consistency proofs.
- **Pick the commitment flavor:**
 - **Default:** IPA/Pedersen Verkle over P-384 or Ristretto255 to keep DLOG-only.
 - **Alternative:** KZG/BLS12-381 if you can run a public, multi-stakeholder ceremony and accept pairings; yields smaller proofs and simpler verifier code.
- **Publish and audit:**
 - **During voting:** periodic roots for VT_cast; after close, publish roots and minimal public witnesses proving the invariants $n_{\text{cast}} = n_{\text{final}} + n_{\text{deleted}}$ and monotonic growth.
 - **Auditor access:** NDA-protected sampled openings for occupancy/counter and supersession links, with privacy-preserving redactions.
- **Integrate with cross-system checks:**
 - Keep your myID/PKIX transaction-count comparison as a statistically sampled, independent corroboration layer. Verkle authenticates IVXV's internal truth; the external logs provide the “second witness” against insider suppression.
- **Prototype, measure, and iterate:**
 - Build a small election-scope pilot with both IPA- and KZG-based backends, measure proof generation latency per 1k updates, witness sizes, and verification load on commodity servers.
 - Run a red-team on revoting leakage and on “count” invariants to ensure no quiet paths to under-/over-count manipulation remain.

If you want, I can sketch concrete data models for VT_cast/VT_final/VT_deleted, the counter/bitmap commitment, and the exact proofs you'd publish at each phase to make the invariants machine-checkable and auditor-friendly without touching IVXV's encryption/mix phases.

ME:

for the first risk, isn't IVXV already has a setup phase to generate the election authority key? they can generate the Verkle "H" values in that same phase. Also, there is already a count value in IVXV that is used as input to the Ballot Processor and then used to prove the integrity of omitting non-counted votes; it is that this count value is just an incrementing counter not cryptographically proved under manipulations.

Copilot:

Trusted setup, “H” generators, and making the counter cryptographic

You're right that IVXV already has a setup phase and a threshold-key ceremony. That gives you a natural operational slot and governance model to generate additional public parameters. But there are two distinct cases hiding under “Verkle setup,” and they have very different trust implications.

Two very different “setups”

- **KZG-based Verkle (pairings, polynomial commitments):**
 - Requires a “powers-of- τ ” ceremony with toxic waste that must be destroyed. If τ leaks, an attacker can forge membership proofs while keeping roots consistent. This is a different failure mode than leaking the ElGamal tally key (which threatens confidentiality, not integrity). You can absolutely run the KZG ceremony in the same calendar phase and with the same multi-party governance, but you should treat it as a separate, auditable ceremony with:
 - A public transcript, many contributors, and verifiability tooling.
 - Clear separation of duties from the decryption-key custodians.
 - Reusability decision: universal setup reused across elections vs. per-election setup.
 - Bottom line: “same phase” is fine; “same ceremony” in the cryptographic sense is not. The assumptions and consequences differ.
- **IPA/Pedersen-based Verkle (inner-product commitments, no pairings):**
 - No toxic-waste ceremony is required. The “H” generator vector can be derived deterministically by hashing a public domain string to curve:
 - Example: $H_i = \text{HashToCurve}(\text{"IVXV-Verkle/H"}, \text{election_id} \parallel i)$, and similarly a base $G = \text{HashToCurve}(\text{"IVXV-Verkle/G"}, \text{election_id})$.
 - Publish the domain string, election_id, curve choice, and derivation function in the setup phase; anyone can recompute and audit that no trapdoor was introduced.
 - Bottom line: you can “generate H” during setup, but it’s a public, reproducible derivation—not a trusted setup.

If your posture is “no new hardness assumptions and no trusted setup,” choose IPA/Pedersen Verkle and derive H deterministically. If you want the smaller proofs and simpler verification of KZG, run a dedicated powers-of- τ with strong optics.

Turning IVXV’s counter into a cryptographic invariant

You already have an operational count that propagates into the Ballot Processor to justify omission of non-counted votes. Today it’s just an incrementing variable—easy to tamper with in theory. You can make it auditable and tamper-evident without touching encryption or mixing.

Here are two robust, simple options that integrate cleanly with a Verkle map of the ballot records:

Option A: Committed counter with per-update consistency

- **State:**
 - Keep the Verkle map VT_cast keyed by salted record IDs (not identities).
 - Maintain a dedicated key c in the same Verkle map holding a Pedersen commitment $C_t = Com(m_t; r_t)$ to the current count m_t .
- **Update rule (on accepting a vote):**
 - Produce a non-interactive proof that $C_{\{t+1\}}$ opens to $m_t + 1$ with some $r_{\{t+1\}}$, and that VT_cast transitioned from $root_t$ to $root_{\{t+1\}}$ with the new record inserted.
 - Publish the new root and the proof tuple (map transition + counter increment). Auditors verify the chain end-to-end.
- **At processing:**
 - Maintain VT_final (eligible i-votes after revotes/paper overrides) and $VT_deleted$ (superseded). Maintain committed counters C_final and $C_deleted$ with proofs that:
 - $m_cast = m_final + m_deleted$
 - Every element moved to $VT_deleted$ corresponds to a supersession link (prove on committed identifiers, not identities).

This preserves your current logic but adds machine-checkable evidence that no insider can silently decrement, “replay,” or roll back the counter without breaking a published proof.

Option B: Occupancy bitmap with a sum proof

- **State:**
 - Commit to an occupancy vector B in the Verkle tree (e.g., a packed bitstring across fixed key slots).
- **Invariant:**
 - Provide a zero-knowledge range/sum proof that $sum(B)$ equals the published count, and that inserts flip bits from 0 to 1 only (append-only semantics).

- **At processing:**
 - Show that the final occupancy sum equals initial occupancy minus the number of supersessions (proved in VT_deleted).

This approach is excellent for sampled audits (“open k bits at random”) while keeping raw records opaque.

Either option upgrades your “just a counter” into a cryptographically bound invariant, and both pair naturally with Verkle’s authenticated map.

Practical guidance on “H” and parameter generation

- **If using IPA/Pedersen Verkle:**
 - Choose a well-audited curve (P-384 aligns with Estonian crypto practice; Ristretto255 is a fast, developer-friendly alternative).
 - Publish a derivation spec: domain separation strings, hash-to-curve function, seed inputs (election_id, version), and the count of generators needed. Auditors can recompute H_i and G precisely—no trapdoor risk.
 - **If using KZG Verkle:**
 - Run a separate, public powers-of- τ with many independent contributions (hardware diversity, streaming proofs, on-camera contributions, etc.).
 - Document the distinct risk: leaking τ lets an attacker forge membership proofs for that setup; it does not help decrypt ballots.
 - Prefer a universal setup reused across elections to amortize ceremony cost, unless your risk model mandates fresh τ each time.
-

How this meshes with performance and privacy

- **Performance:** At IVXV scale ($< 1M$ votes), IPA/Pedersen Verkle updates and multiproofs remain in the “seconds” envelope for batch windows, with verification comfortably sub-second per proof on commodity servers. You don’t need Ethereum-grade throughput. Oberst’s sub-second figures for $n \leq 2^{22}$ are a reassuring upper bound for your environment.
 - **Privacy:** Keep Verkle keys as salted, domain-separated hashes of ballot record IDs controlled by the processor. Publish only roots and structural proofs. For revoting audits, expose counts and linkage proofs over committed identifiers—not raw IDs or timestamps. Bucket or coarsen any time-based cross-checks (e.g., myID/PKIX) to reduce correlation risk.
-

A concrete minimal delta you can deploy

1. Adopt IPA/Pedersen Verkle over P-384. In setup, publish the deterministic H_i/G derivation spec.
2. Add a committed counter key c to the Verkle map; on each accepted vote, atomically:
 - insert the new record,
 - update the counter,
 - emit and store a batched proof for the map transition + counter increment.
3. At processing close, publish roots and proofs for:
 - $n_{\text{cast}} = n_{\text{final}} + n_{\text{deleted}}$,
 - monotonic growth of n_{cast} across epochs,
 - well-formed supersession moves (structure only).
4. Keep your existing Ballot Processor logic and Bulletproofs-based vote-correctness unchanged; Verkle authenticates the collector's evolving state, and the counter is no longer a soft spot.

If later you decide the proof sizes/latency should be even smaller, you can swap the backend to KZG Verkle—after you run a proper powers-of- τ ceremony—without changing the data model or invariants.

If you want, I can draft the exact hash-to-curve derivation for H_i/G , a counter update proof sketch for option A, and the batched publication cadence so it fits IVXV's throughput and logging.

ME:

I believe the simplest and straight forward logic is to use KZG Verkle and whatever risks apply to the election Authority key will apply to Tao as well (unless IVXV team plans to deploy the suggestions of both the paper and the poster together and will make benefit from the common modules)

Copilot:

KZG Verkle in IVXV: yes—with clear eyes about τ versus the tally key

Your instinct is practical: KZG Verkle gives you tiny proofs and fast verification, and IVXV already runs a ceremony. You can absolutely piggyback governance and logistics. Just don't conflate the risks of the election authority's decryption key with the powers-of- τ parameter. They live in different threat planes.

How τ risk differs from the tally key

- Decryption key (ElGamal, threshold): compromise threatens confidentiality. Ballots could be decrypted prematurely or post-facto, but integrity proofs and audit trails remain sound if other components hold.
- KZG τ (structured reference string): compromise threatens integrity. An adversary who learns τ can forge valid-looking openings for arbitrary values against an existing root. That undermines any “counted-as-collected” or “no-deletion” claim anchored in KZG proofs, even if you publish the root frequently.

Direct answer: using KZG is fine if you explicitly accept and manage this integrity risk. Treat τ as a separate, first-class secret with its own ceremony and controls—not “the same as” the tally key.

A clean way to deploy KZG Verkle in IVXV

- Ceremony model
 - Run a public, multi-party powers-of- τ with many independent contributors and a public transcript.
 - Prefer a universal SRS with degree \geq your worst-case vector size (e.g., max tree arity/depth for $< 1\text{M}$ records). Then you don’t need a per-election Phase 2.
 - Separate governance: different custodians and observers than the tally key holders; separate venue and artifacts.
- Curve and library
 - Use BLS12-381, with widely reviewed KZG code. Favor battle-tested implementations that support multi-proofs (FK20) and batch verification.
 - Freeze versions early and commission an external audit; avoid “research-grade” branches.
- Data model (minimal change)
 - Commit to ballot-record identifiers (salted, domain-separated hashes of record IDs) as Verkle leaves.
 - Maintain three roots per phase: VT_cast (all accepted i-votes), VT_final (eligible after overrides), VT_deleted (superseded).
 - Publish periodic roots during voting; publish final roots and tie them to phase logs.
- “Count” as a first-class invariant
 - Either maintain a committed counter leaf updated with each insert, with a proof that it increments by one per accepted record; or
 - Maintain an occupancy bitmap (or per-bucket counts) and publish a proof that $\text{sum}(\cdot)$ equals the public count.
 - At close: prove $n_{\text{cast}} = n_{\text{final}} + n_{\text{deleted}}$, plus well-formed supersession moves (structure only).
- Revoting and privacy

- Keep supersession links at the level of committed identifiers; never expose raw IDs or timestamps publicly.
 - Coarsen or bucket any cross-checks against external logs to avoid correlation.
 - Publishing and verification
 - During voting: publish new root + a small, batched KZG multiproof per epoch (e.g., every second or every N votes).
 - After close: publish the invariant proofs. Verifiers recheck in milliseconds; proof payloads remain small.
-

Where you can share modules across “poster” and “paper”

- If you standardize on BLS12-381 for KZG Verkle, your Bulletproofs-based vote-correctness work still runs on an elliptic curve group (often different). You won’t share group code there, but you can share:
 - Serialization, ASiC-E plumbing, container signing, and audit logging.
 - Hash domain-separation conventions, election_id scoping, and parameter registries.
 - Batch-verification patterns and concurrency scaffolding at the collector.

If you later migrate vote encryption to EC ElGamal, you could collapse more crypto domains; that’s a separate, bigger decision.

Practical guardrails to keep the τ risk acceptable

- Ceremony optics: many contributors, independent hardware, streaming the process, automated transcript verification. Treat it as a public event, not a back-room ritual.
 - Reuse policy: one universal SRS for multiple elections reduces ceremony overhead—but be explicit that a future τ compromise would retroactively weaken all uses. If that’s unacceptable, generate per-election SRS (higher overhead, stronger compartmentalization).
 - Compartmentalization: τ artifacts never touch the tally environment; no overlapping personnel with threshold-key custodians.
 - Belt-and-suspenders: keep your external consistency layer (myID/PKIX transaction counts, RLAs on samples). That gives you an independent witness even if someone questioned the KZG layer.
-

Bottom line

Choosing KZG Verkle is reasonable for IVXV's scale and performance needs. Just frame τ as an integrity-critical parameter that warrants its own public ceremony and governance—parallel to, not “the same as,” the tally key. If you lock that down and pair it with a cryptographically bound counter (or occupancy) and your existing cross-system checks, you get a crisp, auditor-friendly path to “removing insiders’ trust” without overhauling IVXV’s core crypto.

ME:

let's check this fine detail more accurately: leaking Tao would render the Verkle proof useless or forgable, but leaking Tao alone without colluding insiders will not allow editing the votes list

Copilot:

Exactly — that’s an important nuance in the threat model.

If we’re talking about **KZG-based Verkle trees**, the structured reference string parameter τ (“Tao” in your shorthand) is purely a *proof system parameter*, not a signing or encryption key for the ballots themselves.

Here’s the breakdown:

What leaking τ actually does

- **Effect on Verkle proofs:**
 - With τ known, an attacker can fabricate *valid-looking* KZG openings for arbitrary values at arbitrary positions in the committed map.
 - That means they could produce a fake proof that “this leaf has value X” even if the committed root corresponds to something else.
 - In other words, the *binding* property of the commitment is gone — the root no longer authenticates the data.
 - **What it does *not* do by itself:**
 - τ does **not** give the attacker write access to the actual IVXV ballot database or the collector’s append-only log.
 - Without collusion from insiders who can alter the stored state and recompute/publish a new root, the public root values you’ve already published remain unchanged.
 - So if the system is logging and publishing roots in real time (or at least at fixed intervals), an outsider with τ but no insider access can’t retroactively change the votes list in the official system — they can only forge *proofs* about it.
-

Why collusion changes the picture

If an insider with write access to the ballot store colludes with someone who knows τ :

1. They can alter the underlying votes list (add, remove, swap entries).
2. They can recompute the Verkle root to match the altered data.
3. They can produce valid-looking proofs for the altered state that match the new root.
4. To an external verifier, the altered state looks legitimate.

That's why τ is an **integrity-critical** parameter:

- Alone, it enables *forgery of proofs* but not *state changes*.
 - Combined with insider write access, it enables undetectable state tampering.
-

Practical mitigation in IVXV

- **Frequent root publication:** Publish signed roots (with the election authority's signing key) at short intervals to an immutable public log. This way, even if τ leaks later, an attacker can't rewrite history without also breaking the signature or the log's immutability.
 - **Separation of duties:** Keep τ generation and custody completely separate from the tally key and from operational access to the collector.
 - **Post-election verification:** Encourage independent verifiers to store and check roots as they are published, so any later mismatch is caught.
-

If you want, I can map this into a **threat matrix** showing τ leakage vs. insider access vs. public root logging, so you can see exactly which combinations are dangerous and which are harmless. That would make the " $\tau \neq$ tally key" distinction crystal clear for stakeholders.