

DAVINCI: A Decentralized Cryptographic Voting Protocol

Pau Escrich¹, Marta Bellés-Muñoz², Roger Baig¹, and Jordi Piñana³

¹ Polytechnic University of Catalonia, 08034 Barcelona, Spain

² Pompeu Fabra University, 08018 Barcelona, Spain

³ Vocdoni Association, 6300 Zug, Switzerland

pau.escrich@upc.edu, marta.belles@upf.edu, roger.baig@upc.edu,
painan@vocdoni.org

Abstract. We present DAVINCI, a decentralized end-to-end verifiable voting protocol with no trusted coordinator. The protocol is structured as a zero-knowledge (ZK) rollup: the election is a ZK-verified state machine in which every transition, from ballot submission to final tallying, is encoded as a circuit constraint, so that no actor can force the protocol into an invalid state without breaking the underlying proof system. A decentralized set of *sequencers* handles ballot validation and aggregation off-chain, while Ethereum serves as a public bulletin board and a data-availability layer via EIP-4844 blobs. For ballot secrecy, DAVINCI uses threshold ElGamal with distributed key generation; for receipt-freeness, sequencers re-encrypt ciphertexts before storing them, and voters may revote at any time. To make revoting effective against bribery and coercion, DAVINCI introduces *silent revoting*: on every state update, sequencers also re-randomize a random subset of stored ballots, and we quantify how well this hides overwrites among routine refreshes. Our open-source implementation shows that DAVINCI supports large-scale elections with succinct on-chain verification and ballot proofs generated on commodity devices, under the honest-device assumption shared by several deployed verifiable-voting systems.

Keywords: Cryptographic Voting · End-to-End Verifiability · Receipt-Freeness · Zero-Knowledge Proofs · Threshold Homomorphic Encryption · Blockchain · Censorship Resistance

1 Introduction

Electronic voting systems still struggle to balance end-to-end (E2E) verifiability, strong voter privacy, receipt-freeness, censorship resistance, and scalability: voters must be able to verify that their ballots were recorded and that the final tally reflects all valid votes, while no one should be able to produce convincing evidence of how a voter voted. Existing E2E-verifiable solutions either rely on semi-centralized authorities to run the critical infrastructure or, when more decentralized, incur costs that make them hard to deploy at scale.

Public blockchains provide a common time reference, an append-only public ledger, and a programmable execution environment whose trust rests on the network rather than on any operator, but running a voting protocol on an Ethereum Virtual Machine (EVM)-compatible chain is impractical: even the boardroom-voting contract of McCorry et al. [35], which keeps ballots secret but is not receipt-free, costs about 3.3 million gas per voter and was demonstrated with 40 voters, and a dedicated sidechain sacrifices the security that motivates using a blockchain. Receipt-freeness matters more on such a substrate: a briber can escrow a payment that releases on proof of a compliant vote, the *Dark DAO* pattern [19], so it is a first-order requirement.

Our proposal, Decentralized Autonomous Vote Integrity Network with Cryptographic Inference (DAVINCI), addresses both problems as a zero-knowledge (ZK) rollup: a decentralized set of operators called *sequencers* collect ballots off-chain, group them into *batches*, and prove that each batch followed the protocol rules; only the resulting state-root commitment and proof reach Ethereum⁴. Every rule (ballot well-formedness, voter eligibility, state-tree updates, homomorphic tally accumulation, re-randomization of stored ciphertexts, and data availability through EIP-4844 blobs [10]) is a circuit constraint, so no actor can deviate without breaking the proof system, and trust reduces to the soundness of the proof systems (Section 3), the integrity of their setups, and the security of the settlement layer. Ballots are encrypted under a threshold ElGamal key [23] generated by a distributed key generation (DKG) protocol; only a threshold of *key wardens* can decrypt the aggregated tally.

Our contributions are: (i) a specification of elections as ZK-verified state machines, with explicit relations for the voter-side and sequencer-side proofs; (ii) a receipt-freeness mechanism based on threshold ElGamal re-encryption, extended with *silent revoting* and a quantitative bound on an observer’s ability to distinguish overwrites from routine re-randomizations (Lemma 1); (iii) a rollup deployment separating off-chain ballot processing from on-chain settlement, with data availability enforced by binding each state-transition proof to an EIP-4844 blob commitment; and (iv) an open-source implementation [38,39], compared with the recent line of SNARK-based ballot-validity work [29,30].

Section 2 reviews related work, Section 3 introduces the preliminaries and system actors, and Section 4 specifies the protocol. Section 5 states the threat model and analyzes the security properties; Section 6 reports implementation and benchmarks; and Section 7 concludes with limitations and future work. Four short appendices give further detail.

2 State of the Art

Foundations of cryptographic voting. Cryptographic e-voting originated with Chaum’s mix-net construction [14], which achieves anonymity by passing ciphertexts through a cascade of shuffling mixes; verifiable shuffles came later.

⁴ We refer to Ethereum as the settlement layer for concreteness; the protocol applies to any EVM-compatible chain supporting EIP-4844-style blob data availability.

Helios [1] was the first deployed web-based open-audit system; its second version [2] combined ElGamal encryption under a key shared among trustees with a public bulletin board and homomorphic tallying, and it remains a standard reference point for E2E verifiability in practice. Belenios [16] refines this design with formal verifiability proofs and a separate registration authority; the BeleniosRF variant [13] additionally achieves receipt-freeness, but only assuming the voting server is honest with respect to that property. ElectionGuard [6] belongs to the same family, combining homomorphic tallying with threshold guardians and a centralized election administrator that drives the tally. All of these systems require a non-trivial setup of the bulletin board, the tally authority, or both, and their operator set is fixed per election rather than open to participation.

Receipt-freeness and revoting. A receipt-free voting system prevents a voter from producing convincing evidence of their vote, mitigating bribery and simple coercion. Benaloh and Tuinstra [7] gave the first formal notion, and Hirt and Sako [28] showed how to achieve it efficiently by combining homomorphic encryption with re-encryption by the authorities, proven to the voter with designated-verifier proofs. Devillez et al. [21] formalize this approach through traceable receipt-free encryption, and Doan et al. [22] generalize it to multiple ballot-processing servers, removing the single-point-of-failure assumption of earlier non-interactive constructions. These constructions remain server-based: none places the protocol on a public bulletin board with enforced data availability, and the re-encrypting servers are selected in advance rather than drawn from an open set.

ZK-SNARKs in e-voting. A recent line of work applies general-purpose ZK-SNARKs to ballot validity and tally computation. Kryvos [29] uses Groth16 proofs over Pedersen commitments to build a publicly tally-hiding system; Huber et al. [30] establish the feasibility of Groth16 ballot-validity proofs for exponential ElGamal ciphertexts across a wide range of ballot formats, showing that proving cost is dominated by the in-circuit encryption rather than by the ballot rules; and PQKryvos [31] ports this framework to post-quantum assumptions. These works focus on the voter-side and tally-side proofs of otherwise conventional bulletin-board systems. DAVINCI is complementary: it uses the same proof machinery for ballot validity but extends SNARK enforcement across the entire pipeline, including eligibility, state updates, ciphertext re-randomization, and data availability.

Blockchain-based voting. McCorry et al. [35] proposed the first self-tallying voting contract on Ethereum, but placing every vote directly on-chain is expensive, and a voter can prove their vote by revealing their secret voting key. MACI [9] uses ZK proofs to discourage bribery, but its coordinator is a single trusted party that can read individual votes; although later versions improve usability, this assumption remains. Cicada [24] and similar timelock- or threshold-encryption designs keep ballots hidden until the tally, but do not re-randomize stored ciphertexts after submission, so voters can still prove how they voted.

Table 1. Comparison of representative voting systems on formal voter-facing security properties and infrastructural assumptions. ✓: achieved; ~: partial or conditional; ×: not achieved.

System	Ballot privacy	Cast-as-intended	Receipt-freeness	Permissionless ballot processing	Public-ledger settlement
Helios [1]	✓	✓	×	×	×
Belenios [16]	✓	×	×	×	×
BeleniosRF [13]	✓	×	~	×	×
ElectionGuard [6]	✓	✓	~	×	×
MACI [9]	~	×	~	×	✓
Cicada [24]	~	×	×	✓	✓
DAVINCI	✓	×	✓	✓	✓

Position of DAVINCI. Architecturally, DAVINCI is a specialized ZK rollup in the spirit of zkSync [34] rather than a bulletin-board system. A general-purpose rollup would not do: its zkEVM proves execution over public state, whereas DAVINCI’s transition circuit consumes sequencer-private re-encryption randomness, obliges the operator to re-randomize stored ciphertexts, and must let any party, a voter included, produce a transition. Bulletin-board systems such as Helios, Belenios, and ElectionGuard also publish proofs of correct operation at every step; the distinction drawn in Table 1 is that their operator set is fixed per election while DAVINCI’s is permissionless, and that DAVINCI composes every per-step proof into one proof-carrying state machine whose settlement contract rejects any deviation, at the price of its own trust assumptions (Section 7). The table’s properties are ballot privacy, cast-as-intended verifiability, receipt-freeness against an observer of the public board, permissionless ballot processing, and public-ledger settlement (for BeleniosRF, receipt-freeness holds only if the re-randomizing voting server is honest; for ElectionGuard, receipt-freeness rests on the in-person encryption device discarding the ballot nonce, and its Internet-voting mode is Helios-like; for MACI, privacy and receipt-freeness hold only against parties other than the coordinator; for Cicada, individual ballots become public once their time-lock puzzles are solved unless its everlasting-privacy extension is used). DAVINCI is the only system compared that combines receipt-freeness without a trusted server, permissionless processing, and on-chain settlement; its missing cast-as-intended mark is a scoping statement, since second-device mechanisms [36,17] compose with it without changing the on-chain logic (Section 5.3).

3 Preliminaries

ZK-SNARKs. A ZK-SNARK lets a prover convince a verifier that they know a witness ω for a public input x of a relation R without revealing ω ; we assume completeness, knowledge soundness, zero-knowledge, and succinctness (Ap-

pendix D), the last of which keeps on-chain verification within a few hundred thousand gas regardless of batch size.

STARKs and zkVMs. A STARK [5] is a hash-based succinct argument with no trusted setup. A zkVM proves that a program, identified by its code commitment, produced a given output on a given input, so a relation is written as ordinary software. DAVINCI uses a STARK-based zkVM, whose proof is then verified inside a SNARK to obtain a constant-size, zero-knowledge proof for settlement (Appendix D).

Threshold homomorphic encryption. DAVINCI uses threshold ElGamal over an elliptic curve. ElGamal is additively homomorphic on encoded messages, so sequencers can aggregate ballots into a single encrypted tally without decrypting individual votes. It also supports *re-encryption*: adding an encryption of zero, $\text{ReEnc}(c; r') = c + \text{Enc}_{\text{epk}}(0; r')$, yields a fresh ciphertext of the same message, computationally unlinkable to c under the Decisional Diffie–Hellman (DDH) assumption; this is the basis of the receipt-freeness argument of Section 5.2. The secret key is shared among m key wardens through a DKG protocol, so opening the final aggregated tally requires the cooperation of at least t of them.

Data structures and public bulletin board. DAVINCI uses a lean incremental Merkle tree [12] over the SNARK-friendly Poseidon hash [25] as the *census tree*, committing to one $(\text{addr}_{\text{idx}}, \text{weight}_{\text{idx}})$ leaf per eligible voter with only its root on-chain, and a sparse Merkle tree (SMT) [18] of fixed depth D over SHA-256 as the *state tree* (Section 4.1). The Ethereum settlement layer plays the *public bulletin board*: its contracts hold the election commitments (census root, encryption key epk , state root) and accept a transition only with a valid proof and a published data blob.

System actors. The *organizer* sets up the election (census, ballot rules, on-chain parameters) and has no further role; *voters* are the census members; *sequencers*, an open set of operators, collect ballots, verify their proofs, and prove state transitions; *key wardens* (*trustees* [1,16] or *guardians* [6] elsewhere) run the DKG, hold shares of the election key, and jointly decrypt the final tally.

4 The DAVINCI Protocol

An election proceeds in five phases: *setup* (the organizer publishes census root, ballot rules, and election parameters), *key generation* (the key wardens run the DKG and publish epk), the *voting period*, *tallying*, and *finalization*, after which any observer can audit the full history from the on-chain proofs and blobs.

4.1 Election State

The election state is an SMT of depth $D = 64$ whose root R_{S_i} is the on-chain commitment to the state S_i . Keys are D -bit integers, so distinct keys never

share a leaf, and the depth is not a capacity limit: ballot slots are indexed by census position, and the only possible collisions are among vote identifiers (see below). Its key space is partitioned into three disjoint namespaces separated at $\text{cfgMax} = 15$ and $N = 2^{D-1}$:

Configuration ($[0, \text{cfgMax}]$) Reserved low-index leaves store the election process identifier pid , the ballot rules mode , the encryption key epk , and the encrypted tally accumulator acc . The circuit enforces that pid , mode , and epk never change across transitions.

Ballot slots ((cfgMax, N)) Each voter owns a single reserved slot $\text{slot}(\text{idx})$, derived deterministically from the position idx of their leaf in the census tree (its authenticated path and the path length, so no two leaves share a slot), holding the (re-encrypted) ciphertext of their latest ballot. Because the circuit recomputes $\text{slot}(\text{idx})$ from the verified census proof, a ballot can only ever be written to its owner's slot. The voter-to-slot mapping is public: any observer can tell *whether* a voter has cast a ballot, but not its content.

Vote identifiers ($([N, 2^D])$) Each accepted submission inserts a fresh *vote identifier* (a *tracker* in the verifiable-voting literature [15]) at leaf

$$\text{vid} = N + (\text{H}(\text{pid}, \text{addr}, k) \bmod N), \quad (1)$$

where k is the voter's fresh encryption randomness. The circuit enforces that the target leaf is *empty* before insertion; in the rare event of a collision, the voter resamples k^5 . Without k , a vid is unlinkable to addr in the random-oracle model.

4.2 Ballot Casting

A ballot is a vector $b = (v_1, \dots, v_n)$ constrained by the election's *ballot mode* $\text{mode} = (n, v_{\min}, v_{\max}, u, e, s_{\min}, s_{\max})$, defining the choice space

$$C_{\text{mode}} = \{(v_1, \dots, v_n) \mid v_{\min} \leq v_i \leq v_{\max}; u = 1 \Rightarrow v_i \neq v_j \forall i \neq j; s_{\min} \leq \sum_i v_i^e \leq s_{\max}\}. \quad (2)$$

Instantiations of (2) yield single- and multiple-choice (binary v_i , $e=1$), approval, ranking ($u=1$), and quadratic voting ($e=2$ with credit budget $s_{\max}$), in the spirit of the choice spaces of [30], so a single circuit and verification key cover all these formats.

To cast a ballot, the voter fetches $(\text{pid}, \text{mode}, \text{epk})$ from the settlement contract, samples fresh randomness k , encrypts $c = \text{Enc}_{\text{epk}}(b; k)$, derives vid via (1), and proves the relation

$$R_{\text{ballot}} = \{((\text{pid}, \text{mode}, \text{epk}, \text{addr}, \text{weight}, c, \text{vid}); (b, k)) : c = \text{Enc}_{\text{epk}}(b; k) \wedge \text{vid per (1)} \wedge b \in C_{\text{mode}}\}, \quad (3)$$

⁵ By a birthday bound, 10^8 submissions collide somewhere in the 2^{63} -leaf namespace with probability below 0.1% per election, and 10^9 below 6%; a collision costs one resubmission, never a vote.

where the census weight **weight** takes the place of $s_{\max}$ when **mode** leaves it unset: weighted voting is enforced through the ballot's value budget, and the tally applies no multiplier. The voter then signs the vote identifier, $\sigma = \text{Sign}_{\text{sk}}(\text{vid})$, with the ECDSA key behind their census address, and submits the package

$$\text{vote} = (\text{pid}, \text{vid}, c, \pi_{\text{census}}, \pi_{\text{ballot}}, \sigma)$$

to a sequencer of their choice, where π_{census} is the Merkle proof that $(\text{addr}, \text{weight})$ sits at index idx of the census tree. Eligibility and authentication are left out of R_{ballot} and verified by the sequencer, which keeps the voter-side circuit small enough for in-browser proving. The voter keeps k to verify inclusion (Section 5.3).

4.3 Batch Processing and State Transitions

Sequencers settle vote packages in batches of up to N_B submissions. A transition from S_i to S_{i+1} is valid only if accompanied by a proof π_Δ for the state-transition relation, which enforces, for every vote_j in the batch B :

1. *Ballot validity*: $\pi_{\text{ballot},j}$ verifies for the public inputs addr_j , weight_j , c_j , vid_j , and the **pid**, **mode**, and **epk** read from the (proven) configuration leaves of S_i .
2. *Authentication*: σ_j is a valid ECDSA signature on vid_j , and addr_j is derived from the recovered public key.
3. *Eligibility*: $\pi_{\text{census},j}$ verifies against the on-chain census root, yielding the leaf position idx_j and weight weight_j ; the circuit checks addr_j and weight_j against the public inputs of $\pi_{\text{ballot},j}$.
4. *Replay protection*: the leaf at vid_j is empty in the current state, and the circuit inserts vid_j there. Since every accepted submission permanently occupies its vid leaf, no vote package can be applied twice: replaying an earlier package (e.g., to undo a revote) fails the emptiness check, and a package with a fresh vid cannot be forged without the voter's signing key.
5. *Tally update*: the accumulator **acc** is maintained as the exact homomorphic sum of the ciphertexts currently stored in the ballot slots. Let c_j^{old} be the ciphertext currently stored at $\text{slot}(\text{idx}_j)$. If the slot is empty, the ballot is a new submission and the stored ciphertext $\text{ReEnc}(c_j; r_j)$ of step 6 is added to **acc**. Otherwise, the ballot overwrites the previous one: subtract c_j^{old} from **acc** and add $\text{ReEnc}(c_j; r_j)$, so only the voter's most recent ballot contributes to the tally. Each silent refresh (below) also adds its $\text{Enc}_{\text{epk}}(0; r)$ to **acc**, so the equality with the sum of stored ciphertexts is preserved across refreshes.
6. *Re-encryption*: the ciphertext written to $\text{slot}(\text{idx}_j)$ is $\text{ReEnc}(c_j; r_j)$, never the voter's original c_j . The scalars r_j are derived in-circuit from one sequencer-private per-transition seed by a SHA-256 chain, $r_0 = \text{H}(\text{tag} \parallel \text{seed} \parallel R_{S_i})$ for a fixed domain tag and $r_{h+1} = \text{H}(r_h)$, each digest reduced modulo the BN254 scalar-field prime p , and consumed in a fixed order at one scalar per active ciphertext, so no scalar repeats within or across transitions except with negligible probability. The reduction leaves $r_h \bmod \ell$ (ℓ the subgroup order)

$2^{-7.6}$ -close to uniform, so Section 5.2 assumes DDH for this exponent distribution; a 512-bit reduction would close the gap. The seed is the only secret input and is discarded after proving.

Both blob lists of Section 4.4 are canonically sorted by leaf index; the per-ballot processing order inside a batch is fixed as a private witness of the circuit.

Silent revoting. In addition to the ballots in B , the circuit obliges the sequencer to re-randomize ρ further ballot slots that are occupied in the state tree and not written by B . The protocol fixes constants $\tau \geq 1$, $\kappa \geq 0$, $\rho_{\min}$, and $\rho_{\max}$, and enforces $\rho \geq \min(\text{target}, O - w)$ with $\text{target} = \min(\rho_{\max}, \max(\rho_{\min}, \tau w, \kappa|B|))$, where w is the number of overwrites, $|B|$ the batch size, and O the number of occupied slots before the transition; O enters the circuit as a public input that the settlement contract checks against its running count of votes minus overwrites. The refreshed slots are chosen by the sequencer from private randomness, uniformly without replacement among occupied slots the batch does not write (Section 5.2 explains why the selection must not be derivable from public data). The circuit enforces the count rule, that the refreshed slots are distinct occupied ballot slots disjoint from the batch’s writes, that each refresh is a valid re-randomization with a scalar drawn from the chain of step 6, and that the refresh deltas enter acc . The set of slots whose ciphertexts change between R_{S_i} and $R_{S_{i+1}}$ thus mixes genuine overwrites with routine refreshes.

4.4 Data Availability and Liveness

Liveness. Sequencing is permissionless, and settlement runs on a public chain, so progress needs only a live settlement layer and one sequencer willing to produce a valid transition; the failure of individual sequencers does not halt the election.

Data availability. For each transition, the sequencer publishes one or more EIP-4844 blobs with the changed leaves of the state tree: the new vote identifiers, a *single uniform list* of slot updates $(\text{idx}, c^{\text{new}})$ covering new submissions, overwrites, and silent refreshes alike, and the updated accumulator. Which identifier belongs to which slot, and which occupied-slot update is an overwrite, remain private witnesses of the circuit: the blob labels neither, excludes the voter’s original ciphertext, signature, and randomness, and its accumulator is recomputable from the update list, so it carries no information about which updates are overwrites. This suffices to reconstruct the tree while preserving the uncertainty quantified in Lemma 1. The blob encodes the transition as sorted lists of field elements (Appendix A); the circuit builds the cells from its own verified witness, evaluates each blob’s polynomial at a challenge point derived from its KZG commitment, and exposes commitments and evaluations as public inputs, which the settlement contract checks against the blobs’ versioned hashes via the point-evaluation precompile. Thus, sequencers cannot advance the state while withholding the data needed to reconstruct it. EIP-7594 [20] caps one transaction at six blobs; the contract reads all of a transition’s blobs from the transaction that submits it, so the sequencer sizes each batch to fit (Appendix B).

Blobs are pruned from consensus nodes after 4096 epochs (approximately 18 days) [10], but the KZG commitments remain permanently on-chain, so any party can archive blob data and later prove its authenticity against them; long-term auditability thus requires at least one honest archiver (Section 7).

4.5 Tallying and Open Inclusion

Once the voting period closes, the settlement contract stops accepting transitions. The final encrypted tally is the accumulator `acc` committed in the last accepted state. At least t key wardens publish partial decryptions of this single aggregate ciphertext with Chaum–Pedersen proofs of correct decryption, verified on-chain against the committed accumulator; individual ballot ciphertexts are never decrypted.

DAVINCI has no privileged path for individual ballots: a voter cannot submit a vote package to the settlement contract directly, and no sequencer is obliged to process any particular package. Censorship resistance follows instead from the open sequencer set. Any party can run a sequencer, and a valid transition may carry a single vote package, so a voter whose package is withheld by one sequencer submits it to another or processes it themselves: the proof needs only public data (the on-chain configuration and the census) plus the package, and the settlement contract accepts a valid transition from whoever presents it against the current root. Inclusion therefore requires only that some sequencer, possibly the voter’s own, produces a transition before the voting period closes, at the cost of one proof (about 16 s on a consumer GPU, Section 6.1) and one settlement transaction; a colluding sequencer set can delay a ballot but cannot exclude it without excluding every transition. We call this *open inclusion*. Its receipt-freeness implications are analyzed in Section 5.2.

5 Security Analysis

5.1 Threat Model

We assume a computationally bounded adversary that may control any subset of sequencers (including all of them), corrupt fewer than t key wardens, and monitor all public communications, including the Ethereum settlement layer; the adversary cannot break the underlying cryptographic primitives (DDH on the encryption curve, knowledge soundness of the proof systems under honestly generated setups, and the hash functions, modeled as random oracles) or the settlement layer itself. The organizer is trusted only for census formation: a malicious organizer can admit ineligible voters or exclude eligible ones but cannot affect any other property once the census root is published. On the voter side, we adopt the *honest-device assumption*: the casting device encrypts the chosen ballot honestly. Other client-side-encryption protocols [1,16,6,9] make the same assumption or relax it with a cast-as-intended check such as Benaloh challenges; DAVINCI includes none (Section 5.3).

Within this model, DAVINCI targets four guarantees: (i) *integrity*, even against a fully malicious sequencer set; (ii) *ballot secrecy and receipt-freeness*, against fewer than t compromised key wardens and a coercer who reads the bulletin board or is present only intermittently (an adversary who obtains the voter’s signing key, forces abstention, or remains present throughout the voting period is outside our model); (iii) *data availability and liveness* (Section 4.4); and (iv) *censorship resistance*, via open inclusion: the sequencer set is permissionless and a transition may carry a single package (Section 4.5).

Integrity and soundness. Integrity rests on the knowledge soundness of the composed proof system (Section 6): every rule of Section 4.3 is a constraint of the transition relation and the settlement contract rejects any transition whose proof fails, so *state consistency* (every counted vote comes from a census member, with its authenticated weight, in its owner’s slot), the absence of *double voting* (a second submission is an overwrite by construction and step 5 counts only the latest ballot), and *replay protection* (step 4) hold without trusted bookkeeping; batch-internal order is fixed inside the circuit and the contract serializes batches on the current on-chain root. One reordering vector remains: a sequencer can *withhold* a not-yet-applied package and settle it after a later revote, making the older ballot count, since packages carry no creation-time order; the voter detects this through their trackers (the stale identifier appears after the newer one) and recovers by revoting through any sequencer (Section 4.5). *Data withholding* is excluded by the blob binding of Section 4.4.

Ballot secrecy. Ballot processing is publicly attested, yet ballot contents remain hidden: the zero-knowledge property of the ballot and transition proofs ensures that they leak nothing about b or k ; ballots are encrypted under epk , whose secret key is t -of- m shared among the key wardens; and only the final aggregate is ever decrypted, so with at most $t - 1$ compromised wardens no individual ciphertext can be opened. The processing sequencer learns the voter’s address and ciphertext, but not the plaintext; link privacy toward that sequencer is not claimed.

5.2 Receipt-Freeness and Coercion

DAVINCI targets receipt-freeness in the sense of Benaloh and Tuinstra [7] and Hirt and Sako [28]: a voter cannot produce convincing evidence of how they voted to a third party observing the public bulletin board. The coercers covered and excluded are those of Section 5.1. Three mechanisms combine to provide this.

The first is *re-encryption* (step 6 of Section 4.3): the on-chain ciphertext is never the one the voter produced, so revealing (b, k) to a buyer proves nothing about the stored ballot; the circuit enforces it, so even a sequencer colluding with the voter cannot skip it; it gives no unlinkability against the processing sequencer, which saw the submission anyway. The second is *revoting*: only the last valid ballot counts, so a voter coerced at one moment can silently replace

the ballot later, in the spirit of the revoting variants of [16]. The third is *silent revoting*, which protects the revote event itself: without it, a coercer could watch the victim’s (public) slot and detect any later change; with it, slot changes are ambiguous. Two design points follow. The refreshed set must not be a function of public data, or an observer recomputes it and subtracts it from the changed slots, recovering every overwrite; its uniformity is therefore an honest-sequencer assumption, the right trust boundary since the processing sequencer knows the overwrite set anyway, and the circuit enforces only what a colluding sequencer could otherwise skip. And the refresh deltas must enter the accumulator: otherwise Δ_{acc} minus the newly written ciphertexts equals the sum of the overwrite deltas, and a single overwrite is found by a linear scan over the changed slots.

Lemma 1. *Consider a transition with w overwrites and ρ silently refreshed slots drawn from the $O - w$ occupied slots not written by the batch, for a sequencer that samples the refreshed slots uniformly at random without replacement. Under the DDH assumption, for a changed occupied slot chosen by a probabilistic polynomial-time observer of the public bulletin board (including the blob of Section 4.4) without side information about the voter behind that slot, the observer’s posterior probability that the slot was overwritten exceeds the baseline $w/(w + \rho)$ by at most a negligible amount; the bound is a posterior for such a slot, not a classification success rate over the whole batch. Under the protocol constraint $\rho \geq \min(\text{target}, O - w)$ with $\text{target} = \min(\rho_{\max}, \max(\rho_{\min}, \tau w, \kappa|B|))$ and $\rho_{\max} \geq \max(\tau, \kappa)N_B$, the posterior is at most $\min(1/(1 + \tau), w/(w + \kappa|B|))$ outside the saturation regime ($\text{target} < O - w$), and w/O in saturation, where every occupied slot the batch did not write is refreshed and so all O occupied slots change.*

Proof (sketch). By the IND-CPA security of ElGamal (equivalently, DDH), a hybrid argument replaces each re-encrypted and refreshed ciphertext with a fresh encryption of the same plaintext, so the observer’s view differs between the two cases only in *which* slots changed. When the sequencer samples the refreshed subset uniformly at random from the occupied slots the batch did not write, changed slots are exchangeable, and the posterior that a given changed slot is an overwrite equals the prior $w/(w + \rho)$; the constraints $\rho \geq \tau w$ and $\rho \geq \kappa|B|$ give the two arms of the outer minimum. In saturation, every occupied slot changes, and the posterior collapses to w/O . $\square$

The lemma covers a single transition; a coercer correlating observations across transitions is not captured, and a multi-transition analysis is ongoing (Appendix C gives the coverage a slot receives over an election). Our implementation sets $\tau = 2$, $\rho_{\min} = 16$, $\kappa = 1$, and $\rho_{\max} = 2048 = \tau N_B$ with $N_B = 1024$.

Self-sequencing and receipt-freeness. Open inclusion lets a voter process their own package, but whoever operates the sequencer holds the re-encryption seed of that transition (step 6 of Section 4.3) and can therefore prove the content of the stored ciphertext for every ballot it processed. A voter who self-sequences, or who is made to submit through a coercer’s sequencer, thus loses deniability

for that submission, as a Helios voter does. The ballot is still re-encrypted before entering the state tree, and the voter can later revote through another sequencer, restoring deniability for the ballot that actually counts.

5.3 End-to-End Verifiability

Following the standard decomposition [15], we discuss recorded-as-cast, tallied-as-recorded, eligibility, and cast-as-intended verifiability.

Recorded-as-cast. The voter retains k and recomputes vid via (1); any party holding the state tree, which the blobs let anyone rebuild, can then produce a Merkle inclusion proof for the vid leaf against the on-chain state root, which the voter checks locally. This tracker does not undermine receipt-freeness: the leaf proves that a submission was recorded, never its content, and without k third parties cannot link a vid to a voter. A coercer who extracts k can verify that the coerced submission was recorded, but cannot tell whether it was later overwritten, since subsequent submissions carry fresh identifiers unlinkable to the address and slot overwrites are hidden by Lemma 1. A package submitted within the last proving-and-settlement window before the close can miss it, even if the voter processes it themselves. *Tallied-as-recorded.* Universal verifiability follows from the on-chain sequence of transition proofs plus the published blobs: any observer replays the state evolution, verifies the published accumulator value against the proofs, and checks the key wardens’ decryption proofs against the final commitment. *Eligibility.* The transition circuit checks a census Merkle proof for every ballot, so no ballot cast by an ineligible voter contributes to the tally; observers confirm it by verifying π_Δ . *Cast-as-intended.* DAVINCI does not provide it inside the protocol: the honest-device assumption covers the encryption step, and the sequencer-side re-encryption prevents the voter from auditing the stored ciphertext afterward. Second-device audit schemes [36,17] compose naturally, since they operate between voter and casting device before submission and require no change to the on-chain logic.

Everlasting privacy. DAVINCI does not provide everlasting privacy: the voter-to-slot mapping is public, so an adversary who breaks the encryption in the future (e.g., with a sufficiently powerful quantum computer) and has archived the blob data can decrypt individual ballots and link them to voters.

6 Implementation and Benchmarks

To evaluate DAVINCI’s feasibility, we developed an open-source reference implementation [38,39].

Architecture. The pipeline has two proving stages. The ballot circuit [38] is written in Circom [3] and proved in the browser with Groth16 [26] using snarkjs [32] over WebAssembly. The sequencer side [39], the *transition circuit* of the previous sections, is a single program implementing the transition relation of Section 4.3,

compiled to RISC-V and proved in ZisK [37] (v1.3.0-alpha), a STARK-based zkVM (Section 3). It implements every rule of that relation, verifying the N_B ballot proofs in one batched pairing check with in-program Fiat–Shamir coefficients. The STARK is recursively compressed and wrapped into a PLONK proof over BN254, verified on-chain with the BN254 precompiles (EIP-196/197 [11]); only this proof is published, and its zero-knowledge property hides the STARK and hence the sequencer’s witness. The wrapper circuit is fixed by ZisK: the program’s code commitment and the verification key of ZisK’s recursion circuit enter only as public inputs, hashed with the public values and pinned by the settlement contract, so upgrading the transition program changes one stored commitment rather than the verifier or its setup. The ballot circuit’s Groth16 phase-2 CRS is thus the only circuit-specific trusted setup (a multi-party ceremony in the random-beacon model [8], reused across elections); the STARK needs none, and the PLONK wrapper’s setup is universal (the Hermès powers-of-tau [27]).

Curves. BN254 (about 100 bits of security after the Kim–Barbulescu attacks [33]) is forced by the EVM precompiles for the wrapper and also carries the ballot proofs, since ElGamal uses BabyJubjub [4], whose base field is the BN254 scalar field, so encryption is native in the ballot circuit; voters sign with secp256k1 ECDSA; and the blob binding uses the BLS12-381 KZG commitments of EIP-4844. The transition program relies on ZisK precompiles for SHA-256, the BN254 pairing, BabyJubjub, secp256k1, and 256-bit modular arithmetic. The ballot layout is fixed at 16 ciphertext fields ($n \leq 16$); re-encryption skips the unused fields, so per-ballot work grows with the declared n .

The implementation includes the ballot circuit, transition program, sequencer service, and settlement contract (Appendix A); the DKG component is implemented but not yet integrated. It is a research prototype: independent audits of the circuits, program, and contracts, and hardening of the sequencer infrastructure, are required before large-scale deployment.

Data availability and settlement costs. A slot update takes $1 + 2n$ field elements (blob counts in Table 2, encoding in Appendix A); blob fees are priced by a separate fee market and were far below the verification cost in our measurement window. A settlement transaction costs about 498 K gas with one blob plus about 56 K per additional blob, independent of the batch size. On Ethereum mainnet, at the daily-average gas prices of 0.3–5 gwei observed from September 2025 to September 2026⁶ and an ETH price of 1,500–4,000 USD, settling a 128-ballot batch costs roughly 0.2–10 USD at $n=2$ (one blob) and 0.3–12 USD at $n=16$ (three blobs), i.e., *0.002–0.1 USD per settled ballot*, and a 1024-ballot batch at $n=2$ (three blobs, about 612 K gas) 0.0003–0.01 USD per ballot. On Gnosis Chain, whose gas is paid in the USD-pegged xDAI, the same batch costs about

⁶ Daily averages from the Etherscan gas-price chart (<https://etherscan.io/chart/gasprice>); monthly means ranged from 0.6 to 2.0 gwei, and the only day above 5 gwei was 10 October 2025, at 18 gwei.

0.003 USD at the year’s median gas price of 5.2 gwei⁷, 200–600 times less than on mainnet at its median of 0.74 gwei, but a block carries at most two blobs, limiting a transition to 744 ballots at $n=2$ and 121 at $n=16$.

6.1 Performance Benchmarks

Voter proving. Generating a ballot proof in the browser takes about 3 s on a laptop-class PC (AMD Ryzen AI 5 340, 64 GiB RAM, Brave) and about 6.4 s on a smartphone (Google Pixel 7a, Chrome), after a one-time download of the 35.5 MiB proving key and the 6.5 MiB WebAssembly witness generator. For calibration, Huber et al. [30] report under 30 s with a 1 GB proving key for a 50-ciphertext ballot with libsnark on a desktop CPU; our 65 K-constraint circuit encrypts 16 BabyJubjub ciphertexts natively in the BN254 scalar field, covers the choice-space family of (2), derives the vote identifier in-circuit, and proves in a browser. The closest sequencer-side analog is MACI’s coordinator circuits [9], which assume a single trusted coordinator and lack re-encryption, silent revoting, and data-availability binding.

Sequencer proving. Table 2 reports the sequencer’s critical path per batch, from job submission to the verified PLONK proof (including the prover’s own verification), on one machine with a consumer GPU (NVIDIA RTX 5090, 32 GiB) and a 16-core desktop CPU (AMD Ryzen 9 9950X3D, 64 GiB RAM); ballot-proof generation, done by voters, is excluded. The two parameters are the batch size N_B and the ballot field count n of the election’s ballot mode (Section 4.2); a larger N_B amortizes the fixed cost over more ballots, and a higher n costs more re-encryption work per ballot. The reported times are for the steady state: each batch also carries $\rho = N_B$ silent refreshes (Section 4.3), which cost about 1.6 s at $n=2$ and 5 s at $n=16$ for $N_B=128$; the first batch of an election, which has nothing to refresh, takes 28.5 s and 40.7 s, respectively. The batch-independent cost is about 16 s, dominated by recursion, wrapping, and verification (15.6 s measured for $N_B=2$); the cost above it, refresh included, is about 0.11 s per ballot at $n=2$ and 0.23 s at $n=16$ for $N_B=128$ and grows slowly with the batch (0.15 s and 0.34 s at $N_B=1024$) as the census proofs deepen and the prover switches to its reduced-memory mode; the fixed part dominates small batches, so throughput grows with N_B and peaks near 6 votes per second at 512 ballots for $n=2$ and 3.3 at 256 ballots for $n=16$. The implementation caps N_B at 1024, a limit set by host memory (about 41 GB at 1024 ballots in the prover’s reduced-memory mode, used from 512 ballots at a cost of a few percent of speed); the GPU peaks near 30 GiB from 128 ballots up. Blob count grows with the batch: at $n=16$, 512 ballots span nine blobs and 1024 span 17; EIP-7594 caps one transaction at six blobs, so those two rows are proving figures only; a sequencer settling on Ethereum sizes its batches by the field count (Appendix B), at no cost in throughput since the fastest configurations, 512 ballots at $n=2$ and 256

⁷ Blockscout daily averages (<https://gnosis.blockscout.com/stats>); Gnosis’s 1 gwei blob-fee floor adds about 0.0001 USD per blob.

Table 2. Sequencer proving time per batch (with N_B silent refreshes), throughput, and blobs per transition versus batch size N_B and declared field count n .

N_B	$n = 2$			$n = 16$		
	time	votes/s	blobs	time	votes/s	blobs
2	15.6 s	0.1	1	15.7 s	0.1	1
64	22.5 s	2.8	1	30.3 s	2.1	2
128	30.1 s	4.3	1	45.7 s	2.8	3
256	46.2 s	5.5	1	76.7 s	3.3	5
512	84.3 s	6.1	2	164.3 s	3.1	9
1024	171.2 s	6.0	3	363.9 s	2.8	17

at $n=16$, fit. These numbers are obtained under synthetic workloads; evaluation under real electoral conditions (arrival patterns, revoting rates, adversarial sequencers) remains future work.

7 Conclusion

We presented DAVINCI, a decentralized, E2E-verifiable voting protocol implemented as a specialized ZK rollup: elections are ZK-verified state transitions settled on Ethereum, combining public auditability with scalable off-chain execution, while circuit-enforced re-encryption and silent revoting strengthen protection against coercion and vote-selling. To our knowledge, DAVINCI is the first system to combine a universal ballot abstraction, an open sequencer set, and fully proof-enforced election execution within a single rollup architecture.

Limitations. The guarantees of this paper hold under the following assumptions: an honest casting device (no in-protocol cast-as-intended verifiability), fewer than t corrupted key wardens, honestly generated proof-system setups (Section 6), correctness of the not-yet-audited circuits, program, and contracts, at least one honest archiver of blob data beyond the 18-day retention window, a coercer who is neither continuously present nor holding the voter’s signing key, and, for inclusion, some sequencer willing to process a given package before the period closes, which the voter can satisfy by running one. The security arguments in Section 5 are specification-level reductions; a game-based formalization of the composed system is ongoing work.

Future work. Earlier versions of DAVINCI have already been used in small real-world votes. We plan to release a production-ready version, audited and integrated with real-world credential providers, deploy it in real elections, and analyze and publish the resulting data (arrival patterns, revoting rates, sequencer behavior). An important long-term direction is post-quantum migration of the components whose security rests on discrete-logarithm assumptions (the ballot proof, the on-chain wrapper, and the encryption layer), which would also protect long-term ballot privacy against quantum adversaries.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: 17th USENIX Security Symposium (USENIX Security 08). pp. 335–348. USENIX Association (2008)
2. Adida, B., de Marneffe, O., Pereira, O., Quisquater, J.J.: Electing a university president using open-audit voting: Analysis of real-world use of Helios. In: 2009 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE '09). USENIX Association (2009)
3. Bellés-Muñoz, M., Isabel, M., Muñoz-Tapia, J.L., Rubio, A., Baylina, J.: Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing* (2023)
4. Bellés-Muñoz, M., Whitehat, B., Baylina, J., et al.: Twisted Edwards elliptic curves for zero-knowledge circuits. *Mathematics* **9**(23), 3022 (2021)
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, Paper 2018/046, <https://eprint.iacr.org/2018/046> (2018)
6. Benaloh, J., Naehrig, M., Pereira, O., Wallach, D.S.: ElectionGuard: a cryptographic toolkit to enable verifiable elections. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 5485–5502. USENIX Association (Aug 2024)
7. Benaloh, J., Tuinstra, D.: Receipt-free secret-ballot elections (extended abstract). In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing (STOC)*. pp. 544–553 (1994)
8. Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-SNARK parameters in the random beacon model. *Cryptology ePrint Archive*, Paper 2017/1050 (2017), <https://eprint.iacr.org/2017/1050>
9. Buterin, V.: Minimal anti-collusion infrastructure. *Ethereum Research Forum*, <https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413> (2019)
10. Buterin, V., Feist, D., et al.: EIP-4844: Shard blob transactions. *Ethereum Improvement Proposals*, no. 4844 (2022)
11. Buterin, V., Reitwiessner, C.: EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. <https://eips.ethereum.org/EIPS/eip-197> (2017)
12. Cedoor, Plasencia, V.: LeanIMT: An optimized incremental merkle tree. *Tech. rep., Privacy Stewards of Ethereum* (2024), <https://zkkit.org/leanimt-paper.pdf>
13. Chaidos, P., Cortier, V., Fuchsbauer, G., Galindo, D.: BeleniosRF: A non-interactive receipt-free electronic voting scheme. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 2016)*. pp. 1614–1625. ACM (2016)
14. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM* **24**(2), 84–88 (1981)
15. Cortier, V., Galindo, D., Küsters, R., Müller, J., Truderung, T.: SoK: Verifiability notions for e-voting protocols. In: 2016 IEEE Symposium on Security and Privacy (S&P). pp. 779–798. IEEE (2016)
16. Cortier, V., Gaudry, P., Glondou, S.: Belenios: A simple private and verifiable electronic voting system. In: *Foundations of Security, Protocols, and Equational Reasoning. LNCS*, vol. 11565, pp. 214–238. Springer (2019)
17. Cortier, V., Gaudry, P., Goetschmann, Q., Lemonnier, S.: Belenios with cast-as-intended: Towards a usable interface. In: *Electronic Voting – E-Vote-ID 2024. LNCS*, vol. 15014. Springer (2024)

18. Dahlberg, R., Pulls, T., Peeters, R.: Efficient sparse merkle trees: Caching strategies and secure (non-)membership proofs. In: Nordic Conference on Secure IT Systems (NordSec). LNCS, vol. 10014, pp. 199–215. Springer (2016)
19. Daian, P., Kell, T., Miers, I., Juels, A.: On-chain vote buying and the rise of dark DAOs. Hacking, Distributed. <https://hackingdistributed.com/2018/07/02/on-chain-vote-buying/> (2018)
20. D’Amato, F., Dietrichs, A., Ryan, D., Wang, H.W., et al.: EIP-7594: PeerDAS – peer data availability sampling. Ethereum Improvement Proposals, no. 7594 (2024)
21. Devillez, H., Pereira, O., Peters, T.: Traceable receipt-free encryption. In: Advances in Cryptology – ASIACRYPT 2022. LNCS, vol. 13793, pp. 273–303. Springer (2022)
22. Doan, T.V.T., Pereira, O., Peters, T.: Threshold receipt-free single-pass eVoting. In: Electronic Voting – E-Vote-ID 2024. LNCS, vol. 15014. Springer (2024)
23. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985)
24. Glaeser, N., Seres, I.A., Zhu, M., Bonneau, J.: Cicada: A framework for private non-interactive on-chain auctions and voting. Cryptology ePrint Archive, Paper 2023/1473 (2023)
25. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schafneggler, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 519–535. USENIX Association (2021)
26. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology – EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Berlin, Heidelberg (2016)
27. Hermez Network: Hermez Powers of Tau trusted setup ceremony. <https://github.com/iden3/snarkjs#7-prepare-phase-2> (2021), final BN254 powers-of-tau files (powersOfTau28_hez_final)
28. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: Advances in Cryptology – EUROCRYPT 2000. LNCS, vol. 1807, pp. 539–556. Springer (2000)
29. Huber, N., Küsters, R., Krips, T., Liedtke, J., Müller, J., Rausch, D., Reisert, P., Vogt, A.: Kryvos: Publicly tally-hiding verifiable e-voting. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1443–1457. ACM (2022)
30. Huber, N., Küsters, R., Liedtke, J., Rausch, D.: ZK-SNARKs for ballot validity: A feasibility study. In: Electronic Voting – E-Vote-ID 2024. LNCS, vol. 15014. Springer (2024)
31. Huber, N., Reisert, P., Küsters, R.: PQKryvos: Post-quantum secure e-voting with flexible ballot formats and public tally-hiding. Cryptology ePrint Archive, Paper 2026/1004; to appear in *Proceedings on Privacy Enhancing Technologies* (PoPETs) 2026 (2026), <https://eprint.iacr.org/2026/1004>
32. iden3: snarkjs: zkSNARK implementation in JavaScript and WebAssembly. <https://github.com/iden3/snarkjs> (2024)
33. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: Advances in Cryptology – CRYPTO 2016. LNCS, vol. 9814, pp. 543–571. Springer (2016)
34. Matter Labs: zkSync: Scaling and privacy engine for Ethereum. Technical Report, Matter Labs (2020), <https://zksync.io>
35. McCorry, P., Shahandashti, S.F., Hao, F.: A smart contract for boardroom voting with maximum voter privacy. In: Financial Cryptography and Data Security. LNCS, vol. 10322, pp. 357–375. Springer (2017)

36. Müller, J., Truderung, T.: CAISED: A protocol for cast-as-intended verifiability with a second device. In: Electronic Voting – E-Vote-ID 2023. LNCS, vol. 14230. Springer (2023)
37. Polygon Labs: ZISK: a zkVM for general-purpose computation. <https://github.com/0xPolygonHermes/zisk> (2025), version 1.3.0-alpha
38. Vocdoni: davinci-circom: DAVINCI ballot circuit and browser prover. <https://github.com/vocdoni/davinci-circom> (2026), test web application at <https://vocdoni.github.io/davinci-circom>
39. Vocdoni: davinci-zkvm: DAVINCI sequencer on the ZisK zkVM. <https://github.com/vocdoni/davinci-zkvm> (2026)

A Blob Encoding and Settlement Checks

The blob encodes the transition as 32-byte field elements: the count and the sorted list of new vote identifiers, the count and the sorted list of slot updates, each as its index followed by the n active ciphertexts with every curve point compressed to one field element (y plus the parity of x), and the updated accumulator; a transition spans $K = \lceil \text{cells}/4096 \rceil$ blobs. A slot update takes $1 + 2n$ field elements (33 at $n=16$, 5 at $n=2$) and a vote identifier one, so a 128 KiB blob (4096 elements) carries about 120 updates at $n=16$ and about 800 at $n=2$; a 128-ballot batch with its silent refreshes spans one blob at $n=2$ and three at $n=16$, and a 1024-ballot batch three and 17.

For each blob j the circuit derives a challenge point $z_j = H(\text{pid}, R_{S_i}, \text{com}_j)$ from the blob’s KZG commitment com_j , evaluates $y_j = P_j(z_j)$ of the polynomial encoding that blob, and exposes $H(\text{com}_1, y_1, \dots, \text{com}_K, y_K)$ and K as public inputs. The settlement contract verifies the PLONK proof, checks root continuity, the census root, and the occupancy count O against its own counters, recomputes the digest from the submitted commitments and evaluations, and checks each opening (com_j, z_j, y_j) against the blob’s versioned hash via the EIP-4844 point-evaluation precompile. On the simulated backend, a settlement transaction (PLONK verification, the state checks, the digest check, one point evaluation per blob, and the state update) costs about 498 K gas with one blob, and 554 K, 612 K, and 667 K with two, three, and four.

B Blob Capacity and Batch Size

A stored ballot is $2n$ random curve points, one field element each after compression, so its data-availability cost cannot be reduced by encoding; a refresh also publishes a full ciphertext, or an observer could tell it from an overwrite. EIP-7594 [20] caps a transaction at six blobs (24 576 elements) and the settlement contract reads all of a transition’s blobs from the transaction that submits it. A steady-state batch of N_B ballots with $\rho = N_B$ refreshes therefore fits in one transaction iff $2 + N_B + 2N_B(1 + 2n) + 2n \leq 24\,576$, i.e., $N_B \leq \lfloor (24\,574 - 2n)/(3 + 4n) \rfloor$; the sequencer picks the batch size from the election’s n (Table 3). The prover’s own cap is 1024, and the throughput-optimal batches of Table 2 (512 at $n=2$, 256 at $n=16$) fit.

Table 3. Largest batch whose transition settles in one transaction, by ballot field count n (elements per slot update $1 + 2n$; steady state with $\rho = N_B$; first batch of an election with no refreshes; and the resulting batch cap given the prover’s limit of 1024).

n elements/update	steady	N_B	first batch	batch cap
1	3	3510	6143	1024
2	5	2233	4095	1024
4	9	1292	2456	1024
5	11	1068	2047	1024
6	13	909	1754	909
8	17	701	1364	701
12	25	481	944	481
16	33	366	721	366

C Refresh Parameters and Coverage

Lemma 1 bounds what a coercer learns from one transition; how often a given ballot is re-randomized depends on when it was cast. Under $\rho_t = \kappa|B_t|$ and steady arrivals across T transitions, a slot occupied since transition t_0 receives about $\kappa \ln(T/t_0)$ expected refreshes and is never refreshed with probability about $(t_0/T)^\kappa$, so late voters are covered less; a fixed-fraction rule $\rho \geq f \cdot O$ would remove this asymmetry at a cost that grows with the election, and we leave it as a parameter choice. The measured cost of a refresh is about 40 ms of proving per refreshed slot at $n = 16$ and 13 ms at $n = 2$, roughly 12% and 6% of a 128-ballot batch proof at $\kappa = 1$; in the blob-based deployment a refresh publishes the same slot update as a vote, so $\kappa = 1$ doubles the number of slot updates, which the compact encoding of Appendix A keeps within one blob for a 128-ballot batch at $n=2$ and three at $n=16$.

D Proof-System Background

A zero-knowledge succinct non-interactive argument of knowledge (ZK-SNARK) lets a prover convince a verifier that, for a relation R and public input x , they know a witness ω with $(x, \omega) \in R$, without revealing ω . We assume the four standard properties: *completeness* (honest proofs are accepted), *knowledge soundness* (a bounded prover cannot produce a valid proof without knowing such a witness), *zero-knowledge* (the proof leaks nothing about ω), and *succinctness* (constant or logarithmic proof size, with verification far cheaper than re-executing the computation). The last property makes settlement on Ethereum practical, keeping on-chain verification within a few hundred thousand gas regardless of batch size.

A STARK [5] is a succinct argument whose security rests on a hash function alone, in the random-oracle model: it needs no trusted setup and is plausibly post-quantum, but its proofs of tens to hundreds of kilobytes are costly to verify on Ethereum. A *zkVM* is a proof system for a fixed instruction set, whose underlying argument may be a STARK or a SNARK: it proves that a program

identified by its code commitment produced a given output on a given input, so a relation is written as ordinary software. We use a STARK-based zkVM, whose proof we verify inside a SNARK to obtain a constant-size, zero-knowledge proof for settlement.