title

Abstract. Proving the validity of ballots is a central element of verifiable elections. Such proofs can however create challenges when one desires to make a protocol receipt-free.
We explore the challenges raised by validity proofs in the context of protocols where (threshold) receipt-freeness is obtained by secret sharing (an encryption of) a vote between multiple authorities. In such contexts, previous solutions verified the validity of votes by decrypting them after passing them through a mix-net. This approach however creates subtle privacy risks, especially when invalid votes leak structural patterns that threaten receipt-freeness.
We propose a different approach of threshold receipt-free voting in which authorities re-randomize ballot shares then jointly compute a ZK proof of ballot validity before letting the ballots enter a (possibly homomorphic) tallying phase. Our approach keeps the voter computational costs limited while offering verifiability and improving the ballot privacy of previous solutions.
We present two protocols that enable a group of servers to verify and publicly prove that encrypted votes satisfy some validity properties: MiniMix, which preserves prior voter-side behavior with minimal overhead, and HomoRand, which requires voters to submit auxiliary data to facilitate validation over large vote domains. We show how to use our two protocols within a threshold receipt-free voting framework. We provide formal security proofs and efficiency analyses to illustrate trade-offs in our designs.

1 Introduction

Receipt-freeness (RF), is a central property of voting systems that ensures that voters cannot prove how they voted to a third party. Since its formal introduction by Benaloh and Tuinstra [1], RF has been the subject of extensive research, especially because of the tension that it creates with the transparency and verifiability of the election process.

A standard approach first explored by Hirt [11] consists in asking voters to encrypt their vote and submit the ciphertext to a ballot processing server that privately re-randomizes the ciphertext before posting it on a public bulletin-board. Since the re-randomization removes the voter's knowledge of the encryption randomness, voters cannot offer any evidence of the ciphertext content to any third party. In Hirt's solution, the ballot processing server must also send a designated verifier proof that it re-randomized the ballot without modifying its content. Subsequent works have aimed to eliminate the need for sending such a proof while preserving RF using various types of randomizable signatures [2, 3, 5–7]. However, in all these works, receipt-freeness depends on a single trusted ballot processing server: if the ballot processing server leaks the randomness used to re-randomize the ballot, RF is lost (though privacy and verifiability would still be preserved).

In order to remove this single point of failure, Doan et al. [8] introduced a threshold receipt-free voting framework, leveraging multiple ballot processing servers (randomizers) to decentralize trust. Their system ensures RF and correctness as long as the number of corrupted randomizers remains below a threshold. Moreover, it maintains universal verifiability via mixnets.

However, Doan's protocol does not include any ballot validity proof in the ballot submission process: the validity of the votes is only verified when the mixed ciphertexts are decrypted. As a result, a coercer or a vote buyer could ask to see a very unusual invalid vote among the decrypted ballots (it could simply ask to encrypt a large randomly chosen value). This is a limited form of receipt, in the sense that the voter can only demonstrate that he submitted an invalid vote, but it would still be desirable to avoid it whenever possible.

Worst situations may happen when the set of valid votes is very large. For instance, if a vote is by approval among 100 candidates, a voter could simply demonstrate how he voted by submitting a highly unusual approval pattern, which will come uniquely in the tally. This limitation highlights the need for alternative solutions that preserve the security requirements of threshold receipt-freeness while avoiding direct decryption of all individual votes, whether they are valid or not.

Tallying processes based on the homomorphic aggregation of ballots instead of a mixnet can offer a solution to these problems in the context of approval voting (and in other cases), while tally-hiding protocols offer more general solutions: in these protocols, individual votes are never decrypted.

Contributions. We propose a new paradigm for threshold receipt-free voting that addresses a core limitation in [8]: the lack of vote validity enforcement before tallying. Instead of requiring voters to generate complex zero-knowledge proofs at ballot submission time, we defer validity checking to a *pre-tally phase*, executed after combined ciphertexts are reconstructed but before they are tallied.

In particular, each voter secret-shares their vote and encrypts the shares under a traceable receipt-free encryption scheme (TREnc) [5], then submits the resulting encrypted shares (so called ballot shares) to a set of independent ballot processing servers. These servers rerandomize the ballot shares and post them to a public bulletin board. After voting concludes, any entity (e.g., the talliers) can reconstruct each final ciphertext by combining a threshold number of valid ballot shares. These combined ciphertexts are then subject to a pre-tally validation protocol executed in a multi-party computation (MPC) style. Specifically, the talliers, or those who hold shares of the decryption key, collaboratively validate whether each ciphertext corresponds to a vote in the allowed domain, without learning the vote or revealing any auxiliary information. The valid ballots are then pushed into a standard tallying process. This shift enables the enforcement of vote-domain constraints in threshold RF settings, which was previously impractical due to limitations of rerandomization and secret sharing. We realize this paradigm through two protocol instantiations:

– MiniMix, which mirrors the architecture of Doan et al. [8], with voters submitting encrypted shares of their vote under TREnc, and performs MPC-based pre-tally validation over the reconstructed ciphertext.
– HomoRand, which allows voters to submit auxiliary information that facilitates more efficient validity checks in certain settings, while preserving RF.

These constructions extend the threshold RF model of Doan et al. beyond mixnet-based designs to support tallying via homomorphic methods and accommodate arbitrary vote domains $\{v_1, \ldots, v_d\}$, thus broadening applicability beyond binary tallies. While HomoRand achieves asymptotically better pre-tally efficiency, with cost scaling logarithmically in the domain size d, it introduces higher voter-side overhead. In contrast, MiniMix imposes minimal computational burden on voters, making it preferable when client efficiency is critical or d is moderate.

In contrast to general-purpose frameworks for collaborative zero-knowledge (ZK), such as that of Ozdemir and Boneh [13], which adapt ZK-SNARKs to settings with distributed witnesses among cooperative provers, our focus is on designing more direct protocols for voting-specific statements. In our setting, the witness originates from a potentially malicious voter, and properties like receipt-freeness and input validity must be enforced jointly without revealing the vote.

Remark. Although our protocols are motivated by the threshold RF setting of [8], the pre-tally validation phase we propose operates independently of that application, making it more broadly applicable. For instance, one could remove the 0/1 proofs from the CGS97 protocol [4] and insert a round of MiniMix to enforce ballot validity. Such a substitution would functionally ensure well-formed ballots, shifting the computational efforts from voting clients to servers.

2 Building Blocks

Secret Sharing. We use a standard (n, t)-threshold secret sharing scheme [14], consisting of two algorithms. The sharing algorithm $\mathsf{Share}(n, t, m)$ splits a secret m into n shares $(m_1, \ldots, m_n)$ such that any subset of at least t shares can reconstruct m, while any set of fewer than t shares reveals no information about m. The reconstruction algorithm $\mathsf{Combine}(n, t, m_1, \ldots, m_t)$ uses Lagrange interpolation to recover m from any t shares.

Traceable Receipt-Free Encryption (TREnc). We recall the notion of TREnc [5], a public-key encryption primitive that enables the receipt-free submission of secret ballots.

Definition 2.1. *A* TREnc [5] *is a* CPA *encryption* (Gen, Enc, Dec) *with a public encryption key* pk *and secret key* sk*, which is augmented with a 5-tuple of algorithms* (LGen, LEnc, Trace, Rand, Ver)*:*

$\mathsf{LGen}(\mathsf{pk})$*: The link generation algorithm takes as input a public key* pk *in the range of* Gen *and outputs a link key* lk.

$\mathsf{LEnc}(\mathsf{pk}, \mathsf{lk}, m; r)$*: The linked encryption algorithm takes as input a pair of public/link keys* $(\mathsf{pk}, \mathsf{lk})$*, a message* m*, and a randomness* r*, outputs a ciphertext.*

$\mathsf{Trace}(\mathsf{pk}, \mathsf{CT})$ *: The tracing algorithm takes as input a public key* pk*, a ciphertext* CT *and outputs a trace* τ*. We call* τ *the trace of* CT.

$\mathsf{Rand}(\mathsf{pk}, \mathsf{CT})$*: The randomization algorithm takes as input a public key* pk *and a ciphertext* CT*, outputs another ciphertext.*

$\mathsf{Ver}(\mathsf{pk}, \mathsf{CT})$*: The verification algorithm takes as input a public key* pk*, a ciphertext* CT *and outputs* 1 *if the ciphertext is valid,* 0 *otherwise.*

Informally, a TREnc scheme is *traceable* if no efficient adversary can produce a second ciphertext with the same trace as a given ciphertext, yet which decrypts to a different message. It is *TCCA-secure* (Traceable Chosen-Ciphertext Attack secure) if re-randomized ciphertexts are indistinguishable from fresh ciphertexts with the same trace, even in the presence of a restricted decryption oracle.

We assume that it is straightforward to extract specific parts of the TREnc's ciphertext CT. In particular:

$\mathsf{Strip}(\mathsf{pk}, \mathsf{CT})$**:** Returns the CPA-encryption component of CT. For instance, in TREnc [5], this component is $\boldsymbol{c} = (c_0, c_1, c_2) = (g^m f^\theta, g^\theta, h^\theta)$, where m is the message, $\theta \leftarrow\!\!\$\ \mathbb{Z}_p$, and $(f, g, h) \in \mathsf{pk}$.

$\mathsf{CPA}(\mathsf{pk}, m; r)$**:** Computes the CPA component of a TREnc ciphertext on message m with randomness r under TREnc's public key pk. We note that this algorithm can be used independently of TREnc.

We further assume that the underlying CPA encryption is additively homomorphic over the message space. That is, given two ciphertexts $\boldsymbol{C}_1 = \mathsf{CPA}(\mathsf{pk}, m_1; r_1)$ and $\boldsymbol{C}_2 = \mathsf{CPA}(\mathsf{pk}, m_2; r_2)$, we can compute $\boldsymbol{C}_1 \cdot \boldsymbol{C}_2 = \mathsf{CPA}(\mathsf{pk}, m_1 + m_2; r_1 + r_2)$ (up to group operation notation). In many cases, we omit the randomness when it is sampled uniformly at random. In the rest of the paper, we use the prefix TREnc. to indicate that an algorithm belongs to the TREnc suite.

Non-Interactive Proofs. Informally, a proof for an NP relation R is a protocol by which a prover P convinces a probabilistic polynomial-time (PPT) verifier V that $\exists w : R(w, x) = 1$, where x is a called a statement, and w a witness for x. When the proof comprises a single message from P to V, it is said to be non-interactive and has syntax:

– $\mathsf{PrfSetup}(1^\lambda) \rightarrow \sigma$: On input a security parameter λ, outputs a common reference string (CRS) σ;
– $\mathsf{PrfProve}(\sigma, x; w) \rightarrow \pi/\bot$: On input a CRS σ, a statement x, and a witness w, if $R(x, w) = 1$, outputs a proof π, otherwise $\bot$;
– $\mathsf{PrfVerify}(\sigma, x, \pi) \rightarrow \{0, 1\}$: On input σ, a statement x, and a proof π, outputs accept or reject.

A non-interactive zero-knowledge proof of knowledge (NIZKPoK) is a proof that satisfies completeness, knowledge soundness, and zero-knowledge [9].

3 Verifiable Ciphertext Validity in MPC

We present two MPC-based constructions for verifying whether a ciphertext $\boldsymbol{C} = \mathsf{CPA}(\mathsf{PK}, m)$, under a CPA-secure encryption scheme with the corresponding decryption $\mathsf{Dec}(\mathsf{SK}, \boldsymbol{C}) = m$, encrypts a message m satisfying a given constraint (e.g., $m \in \{v_1, \dots, v_d\}$). The protocol reveals only the outcome of the check and leaks no additional information about the underlying plaintext or any related value, even in the case of failure. In the following, we describe two such approaches in detail.

MPC-MiniMix Protocol. The first construction, refereed to as $\mathsf{MiniMix}$ (Algorithm 1), employs an OR-proof mechanism to verify message validity. The core idea is to check whether any of the ciphertexts $\boldsymbol{C}_j = \mathsf{CPA}(\mathsf{PK}, m - v_j)$ encrypts the value 0, where each $\boldsymbol{C}_j$ is derived as $\boldsymbol{C} \cdot \mathsf{CPA}(\mathsf{PK}, v_j)^{-1}$ for $j \in [d]$, exploiting the additive homomorphism of the CPA scheme. Conceptually, this construction generalizes plaintext equivalence proofs [12] to enable a form of privacy-preserving range verification.

The sender first encrypts the message m, from which all parties can compute the ciphertexts $\{\boldsymbol{C}_j\}_{j\in[d]}$. The parties then engage in a collaborative verification process, structured as a simplified mixnet operating over the d ciphertexts. Each party T_i, *in turn*, shuffles the ciphertexts (lines 5–10), with the output of T_k serving as the input to T_{k+1}, for all $k \in [T - 1]$, where T denotes the number of parties. Each shuffle includes a re-randomization step (line 8), ensuring that—under honest execution—the ciphertext encrypting 0 becomes unlinkable to its initial position, thereby preserving zero-knowledge.

A significant challenge arises when the message m is a carefully crafted invalid value chosen. In such cases, decryption may reveal not only that the range verification failed but also partial information about m, specifically $m - v_j$ for some j. This could potentially assist the adversary in extracting structural patterns and re-identify the voter. To mitigate this risk, we incorporate a masking step following the shuffle: after shuffling, each party T_k raises each ciphertext to a fresh random exponent $\alpha_j^{(k)}$ (lines 11–15), and proves correctness via a ZKPoK $\boldsymbol{\pi}^{(k)}$. These ciphertexts and proofs are then broadcast (line 16).

We assume an honest majority setting. If the majority finds that a proof fails verification (line 17), the corresponding party T_k is excluded from the protocol, and the output of the last honest party is used to proceed. Although the shuffling and exponentiation phases are presented separately for clarity and to support later comparative analysis (Section 5.4), they can be efficiently merged into a single phase with a unified zero-knowledge proof. Finally, the parties jointly decrypt the resulting ciphertexts (line 22). If the message m is valid, at least one ciphertext will decrypt to 0 while being different of $\mathsf{CPA}(\mathsf{PK}, 0; 0)$; otherwise all decrypted values appear as random group elements, preventing any information leakage about m.

MPC-HomoRand Protocol. In this approach, we leverage pairings to ensure that the encrypted message m lies within the intended domain $\{v_1, \dots, v_d\}$. Assume without loss of generality that $v_1 \le v_2 \le \dots \le v_d$, and let l be the

Algorithm 1 MPC-MiniMix: Sequential Multi-Party Randomization

procedure MPC-MiniMix$(\mathsf{PK}, \mathsf{SK}, \boldsymbol{C} = \{\boldsymbol{C}_j\}_{j \in [d]})$
  Let $\mathcal{T} = \{T_1, \ldots, T_T\}$ be the set of participating MPC parties
  Initialize $\boldsymbol{C}^{(0)} \leftarrow \boldsymbol{C}$ ▷ i.e., $\boldsymbol{C}_j^{(0)} \leftarrow \boldsymbol{C}_j$ for $j \in [d]$
  for $k = 1$ to T **do** ▷ Each party acts sequentially
    Sample at random a permutation P_k of $\{1, \ldots, d\}$
    for $j = 1$ to d **do** ▷ Shuffle the ciphertexts
      $s_j^{(k)} \leftarrow\$ \mathbb{Z}_p \setminus \{0\}$
      $\boldsymbol{C}'^{(k)}_{P_k(j)} \leftarrow \boldsymbol{C}_j^{(k-1)} \cdot \mathsf{CPA}(\mathsf{PK}, 1_G; s_j^{(k)})$
    end for
    $\boldsymbol{\pi}_{sf}^{(k)} \leftarrow \mathsf{PrfProve}(\mathsf{PK}, \{\boldsymbol{C}'^{(k)}_j, \boldsymbol{C}_j^{(k-1)}\}_{j \in [d]}; P_k, \{s_j^{(k)}\}_{j \in [d]})$
    for $j = 1$ to d **do** ▷ Apply random factors
      Sample random values $\alpha_j^{(k)}, r_j^{(k)} \leftarrow\$ \mathbb{Z}_p \setminus \{0\}$
      $\boldsymbol{C}_j^{(k)} \leftarrow (\boldsymbol{C}'^{(k)}_j)^{\alpha_j^{(k)}} \cdot \mathsf{CPA}(\mathsf{PK}, 1_G; r_j^{(k)})$
      $\boldsymbol{\pi}_{rd}^{(k)} \leftarrow \mathsf{PrfProve}(\mathsf{PK}, \boldsymbol{C}'^{(k)}_j, \boldsymbol{C}_j^{(k)}; \alpha_j^{(k)}, r_j^{(k)})$
    end for
    Publish $\boldsymbol{C}^{(k)} = \{\boldsymbol{C}_j^{(k)}\}_{j \in [d]}$ and $\boldsymbol{\pi}^{(k)} = (\boldsymbol{\pi}_{sf}^{(k)}, \{\boldsymbol{\pi}_{rd}^{(k)}\}_{j \in [d]})$
    if $\mathsf{PrfVer}(\boldsymbol{\pi}^{(k)}) = 0$ **then** ▷ Verify the proofs
      return $\perp$ ▷ Abort if proof verification fails
    end if
  end for
  for $j = 1$ to d **do**
    $m_j \leftarrow \mathsf{Dec}(\mathsf{SK}, C_j^{(T)})$ ▷ Decrypt the final ciphertexts
    if $m_j = 0$ **then**
      return 1 ▷ Return 1 if any decrypted value is 0
    end if
  end for
  return 0 ▷ Return 0 if no ciphertext decrypts to 0
end procedure

smallest number of bits such that the entire range from v_1 to v_d fits within $[0, 2^l - 1]$; that is, $\{v_1, \ldots, v_d\} \subseteq [0, 2^l - 1]$. To prove that $m \in \{v_1, \ldots, v_d\}$, it suffices to verify that both $m - v_1 \in [0, 2^l - 1]$ and $v_d - m \in [0, 2^l - 1]$, which guarantees that m lies within the range bounded by v_1 and v_d. For simplicity, here we describe how to prove that a value $m \in [0, 2^l - 1]$. This is accomplished by having the sender decompose m into an l-bit string $b_1 b_2 \ldots b_l$, encrypting each of them separately in two distinct source groups $\mathbb{G}$ and $\hat{\mathbb{G}}$. Subsequently, a MPC protocol operates in the target group $\mathbb{G}_{\mathcal{T}}$ to jointly verify that each encrypted bit b_j satisfies $b_j \in \{0, 1\}$ for all $j \in [l]$. This binary decomposition allows the sender's computational effort to scale logarithmically with d, i.e., $O(\log d)$.

More precisely, the sender encodes each bit $b_j \in \{0, 1\}$ by encrypting G^{b_j} and $\hat{G}^{b_j}$ under public keys PK_1 and PK_2, respectively, yielding ciphertexts $\boldsymbol{c}_j = \mathsf{CPA}(\mathsf{PK}_1, G^{b_j}) \in \mathbb{G}$ and $\hat{\boldsymbol{c}}_j = \mathsf{CPA}(\mathsf{PK}_2, \hat{G}^{b_j}) \in \hat{\mathbb{G}}$. The parties then jointly evaluate the expression

$$e(G, \hat{G})^{\sum_{j \in [l]} \gamma_j b_j (1 - b_j)} \stackrel{?}{=} 1_{\mathbb{G}_{\mathcal{T}}},$$

Algorithm 2 MPC-HomoRand.Part1: Randomization by Party T_k

procedure $\mathsf{MPC\text{-}HomoRand.Part1}(\mathsf{PK}_1, \mathsf{PK}_2, \boldsymbol{C})$ ▷ Input: $\boldsymbol{C} = \{(\boldsymbol{c}_j, \hat{\boldsymbol{c}}_j)\}_{j\in[l]}$
  for $j = 1$ to l **do**
    $\gamma_{k,j}, \lambda_{k,j}, r_{k,j}, s_{k,j} \leftarrow\!\!\$\ \mathbb{Z}_p \setminus \{0\}$
    $\boldsymbol{c}''_{k,j} \leftarrow \boldsymbol{c}_j^{\gamma_{k,j}} \cdot \mathsf{CPA}(\mathsf{PK}_1, G^{\lambda_{k,j}}; r_{k,j})$ ▷ Apply random factors to $\boldsymbol{c}_j$
    $\hat{\boldsymbol{c}}''_{k,j} \leftarrow (\mathsf{CPA}(\mathsf{PK}_2, \hat{G}; 0)/\hat{\boldsymbol{c}}_j)^{\lambda_{k,j}} \cdot \mathsf{CPA}(\mathsf{PK}_2, 1_{\hat{\mathbb{G}}}; s_{k,j})$ ▷ Apply random factors to $\hat{\boldsymbol{c}}_j$
    $\boldsymbol{\pi}_{k,j} \leftarrow \mathsf{PrfProve}(\mathsf{PK}_1, \mathsf{PK}_2, \boldsymbol{C}, \boldsymbol{c}''_{k,j}, \hat{\boldsymbol{c}}''_{k,j}; \gamma_{k,j}, \lambda_{k,j}, r_{k,j}, s_{k,j})$
    $\boldsymbol{C}''_{k,j} \leftarrow (\boldsymbol{c}''_{k,j}, \hat{\boldsymbol{c}}''_{k,j}, \boldsymbol{\pi}_{k,j})$
  end for
  return $\boldsymbol{C}''_k = \{\boldsymbol{C}''_{k,j}\}_{j\in[l]}$
end procedure

where $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_{\mathcal{T}}$ denotes a bilinear pairing, and each $\gamma_j \leftarrow\!\!\$\ \mathbb{Z}_p$ is a blinding factor. This check succeeds if and only if all bits are well-formed, i.e., $b_j \in \{0,1\}$ for every $j \in [l]$. Crucially, the use of masking randomness ensures that the check does not reveal the index or value of any invalid bit, thereby preserving privacy even in the case of malformed inputs. In the MPC setting, the blinding factors γ_j are additively shared across parties: each party T_k locally samples $\gamma_{k,j} \leftarrow\!\!\$\ \mathbb{Z}_p$, and $\gamma_j = \sum_{k\in[T]} \gamma_{k,j}$. These values are generated *in parallel* across all parties.

We instantiate this technique in the MPC-HomoRand protocol, specified in Algorithms 2 and 3, using a concrete CPA encryption scheme. Let $\mathsf{PK} = (G, H, f) \in \mathbb{G}^3$, with secret key $\mathsf{SK} = (\alpha, \beta) \in \mathbb{Z}_p^2$ and $f = G^\alpha H^\beta$. Encryption of a message $m \in \mathbb{Z}_p$ with randomness r proceeds as $\boldsymbol{c} = \mathsf{CPA}(\mathsf{PK}, m; r) = (c_0, c_1, c_2) = (G^m f^r, G^r, H^r)$, and decryption yields $\mathsf{Dec}(\mathsf{SK}, \boldsymbol{c}) = c_0 \cdot c_1^{-\alpha} \cdot c_2^{-\beta} = G^m$.

In Algorithm 2, each party independently masks the ciphertexts in parallel and proves correctness via a ZKPoK $\{\boldsymbol{\pi}_{k,j}\}_{j\in[l]}$, which are broadcast alongside the resulting ciphertexts. Algorithm 3 specifies the subsequent verification phase: each proof is validated (line 4), and any party that fails verification is excluded. The combination of valid ciphertexts (lines 8–9) is computed solely from the outputs of honest parties, and the final values are jointly decrypted (line 17). The values (a_j, b_j, c_j) computed in lines 12–14 are used to securely mask the term $b_j(1 - b_j)$ which would otherwise leak information if $b_j \notin \{0,1\}$.

Although the description employs a specific CPA encryption scheme for concreteness (see Appendix C.3 for a proof of correctness), our construction generalizes to any additive homomorphic CPA scheme. The core idea, securely verifying bit decomposition without leakage, remains applicable in broader cryptographic contexts.

4 Threshold Receipt-Free Voting

In this section, we adopt the general voting system model of Doan et al. [8], which supports multiple ballot processing servers and enables both threshold receipt-freeness and threshold correctness (Section 4.1). We then review the first threshold receipt-free scheme from [8] (Section 4.2), whose limitations motivate our new constructions in Section 5.

Algorithm 3 MPC-$\mathsf{HomoRand.Part2}$: Multi-Party Verification

procedure MPC-$\mathsf{HomoRand.Part2}(\mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2, \boldsymbol{C}, \{\boldsymbol{C}''_k\}_{k\in[T]})$
  for $j = 1$ to l **do**
    for $k = 1$ to T **do**
      if $\mathsf{PrfVer}(\mathsf{PK}, \boldsymbol{C}''_{k,j}, \boldsymbol{\pi}_{k,j}) = 0$ **then**
        return $\perp$
      end if
    end for
    $\boldsymbol{c}''_j = (c''_{0j}, c''_{1j}, c''_{2j}) \leftarrow \prod_{k=1}^{T} \boldsymbol{c}''_{k,j}$ ▷ Aggregate the ciphertexts
    $\hat{\boldsymbol{c}}''_j = (\hat{c}''_{0j}, \hat{c}''_{1j}, \hat{c}''_{2j}) \leftarrow \prod_{k=1}^{T} \hat{\boldsymbol{c}}''_{k,j}$
    $X_j \leftarrow \mathsf{Dec}(\mathsf{SK}_1, \boldsymbol{c}''_j)$ ▷ Decrypt to obtain the randomized value
    $\bar{\boldsymbol{c}}'_j = (\bar{c}'_{0j}, \bar{c}'_{1j}, \bar{c}'_{2j}) \leftarrow \mathsf{CPA}(\mathsf{PK}_2, \hat{G}; 0)/\hat{\boldsymbol{c}}_j$
    $a_j \leftarrow e(X_j, \bar{c}'_{0j})/e(G, \hat{c}''_{0j})$
    $b_j \leftarrow e(X_j, \bar{c}'_{1j})/e(G, \hat{c}''_{1j})$
    $c_j \leftarrow e(X_j, \bar{c}'_{2j})/e(G, \hat{c}''_{2j})$
  end for
  $(a, b, c) \leftarrow (\prod_{j=1}^{l} a_j, \prod_{j=1}^{l} b_j, \prod_{j=1}^{l} c_j)$
  $Y \leftarrow \mathsf{Dec}(\mathsf{SK}_2, (a, b, c))$ ▷ Decrypt the aggregated result
  if $Y = 1_{\mathbb{G}_\mathcal{T}}$ **then**
    return 1
  else
    return 0
  end if
end procedure

Definition 4.1 (Voting System [8]). *A voting system with n ballot processing servers consists of algorithms:* $(\mathsf{SetupElection}, \mathsf{Vote}, \mathsf{ProcessBallot}, \mathsf{TraceBallot}, \mathsf{Valid}, \mathsf{Append}, \mathsf{Publish}, \mathsf{VerifyVote}, \mathsf{Tally}, \mathsf{VerifyResult})$*, and a result function $\rho_m : \mathbb{V}^m \cup \{\perp\} \to \mathbb{R}$, where $\mathbb{V}$ is the vote domain and $\mathbb{R}$ is the result space.*

The election is initialized via $\mathsf{SetupElection}$. Each voter runs Vote to generate a ballot consisting of n shares, one per ballot processing server. Each share is independently randomized by a distinct server using $\mathsf{ProcessBallot}$. A tracking code is derived via $\mathsf{TraceBallot}$, enabling voters to later trace their ballots on the public bulletin board PBB. Validated shares are collected and appended to the ballot box BB using Append, then made publicly available on PBB via $\mathsf{Publish}$. Voters verify correct recording using $\mathsf{VerifyVote}$ and their tracking codes. The tally is computed over valid ballots, checked by Valid, using Tally, and its correctness is publicly verifiable via $\mathsf{VerifyResult}$.

4.1 Security definitions

Following Doan et al. [8], we recall receipt-freeness and correctness against adversaries that may corrupt up to $\mathsf{t_{rf}}$ out of n ballot processing servers.

Threshold Receipt-Freeness. The definition of threshold receipt-freeness proceeds in two parts. The first ensures that no adversary–despite corrupting up to $\mathsf{t_{rf}}$ out of n ballot processing servers–can coerce a voter into producing a receipt that convincingly proves how they voted. This guarantee holds even if the adversary learns the voter's randomness or attempts to bias the ballot construction. The second part, extends the indistinguishability-based receipt-freeness notion of Devillez et al. [5] to the threshold setting. It formalizes the requirement that even if a voter colludes with up to $\mathsf{t_{rf}}$ servers and deviates arbitrarily from the honest ballot distribution, a third party, given full knowledge of how the ballot was constructed and access to the public bulletin board, should not be able to determine whether the voter submitted that ballot or a different one encoding another vote.

Threshold Correctness. Threshold correctness ensures that the choices of honest voters are faithfully reflected in the final tally, even in the presence of limited adversarial control over the ballot processing servers. Concretely, it guarantees that as long as at least $n - \mathsf{t_{corr}}$ processed shares of each honestly generated ballot appear on the public bulletin board–regardless of whether they were handled by malicious servers–the election outcome remains uniquely determined. That is, the adversary should not be able to construct two sets of valid-looking ballot boxes that lead to different outcomes.

4.2 The Doan et al.'s Voting Scheme

The first protocol to simultaneously achieve threshold receipt-freeness and correctness was proposed by Doan et al. [8] (E-Vote-ID 2024). Their construction, illustrated in Figure 1, introduces a threshold receipt-free voting system that significantly reduces trust assumptions on ballot randomizers.

In their scheme, each voter secret-shares their vote, encrypts each share using a TREnc (Section 2), and submits the resulting ballot shares $\{\mathsf{b}_i\}_{i\in[n]}$ to a set of independent randomizers. Each randomizer then rerandomizes one ballot share and output b'_i made available on the public board BB. After the voting phase,

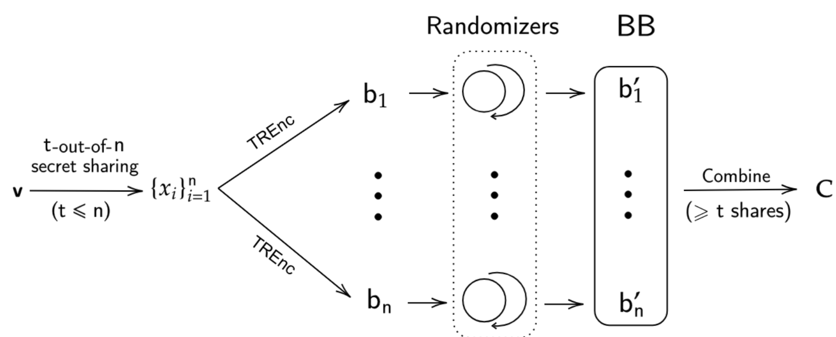

Fig. 1: Doan et al.'s voting scheme.

the talliers (or any observer) extract standard CPA components from the TREnc outputs, and use Lagrange interpolation to reconstruct an encryption of the original vote. This process results in a final ciphertext $\boldsymbol{C}$ that is then submitted into a mixnet-based tallying process. Receipt-freeness is preserved as long as at least one honest rerandomized share b'_i is used in the reconstruction of $\boldsymbol{C}$.

Limitations. Crucially, the protocol does not guarantee that $\boldsymbol{C}$ is actually an encryption of a valid vote. As a result, a maliciously constructed $\boldsymbol{C}$ may encode invalid vote that, upon decryption, leaks unintended information, potentially enabling voter re-identification and violating receipt-freeness. While TREnc provides strong privacy guarantees, it offers limited support when it comes to verifying election-specific constraints on the encrypted vote, which typically require non-linear proofs. For instance, proving that a vote v is a bit typically involves demonstrating that $\mathsf{v}(1 - \mathsf{v}) = 0$, which is a quadratic relation and requires a non-linear proof (see, e.g., [7]).

Although non-interactive zero-knowledge (NIZK) proofs can, in principle, support such constraints, integrating them into this setting with the secret sharing and rerandomization is non-trivial. In particular, a non-linear proof constructed before randomization would not survive the transformation, and constructing such a proof after randomization is infeasible for the voter, who no longer has control over the randomness in the ciphertexts. In a multi-randomizer setting, where multiple servers independently rerandomize different shares of a ballot, adapting a non-linear proof locally is also challenging. The transformation applied by each server generally depends on the randomness chosen by others, rendering any isolated adaptation incorrect or unverifiable. As a result, constructing such proofs typically requires interaction between the voter and all randomizing parties. This interaction not only introduces complexity and performance overhead, but also opens the door to adversarial behavior: malicious servers may collude with a voter to help them construct a verifiable proof of their vote and undermine receipt-freeness. To enhance both the security and practicality of the system, we aim to eliminate this interaction altogether. In the next section, we introduce a new technique that enforces non-linear constraints on encrypted votes without requiring any coordination between the voter and the randomizing servers.

5 Voting Schemes

We propose a new approach that avoids requiring voters to construct complex zero-knowledge proofs at ballot submission time. Instead, we shift the validity checking to a dedicated *pre-tally phase*, which takes place after the final combined ciphertexts $\boldsymbol{C}$ have been reconstructed (see Figure 1), but before tallying begins.

At a high level, our scheme preserves the overall voting flow of Doan et al. from the voter's perspective. The key difference lies in how the reconstructed ciphertexts $\boldsymbol{C}$ are handled. Rather than passing them directly to a mixnet, they are first processed in a multi-party computation (MPC) protocol that verifies whether each ciphertext encodes a valid vote. This verification is performed collaboratively by the talliers or those who hold shares of the decryption key without revealing the vote or any auxiliary information. Only ciphertexts that pass this validation step are forwarded to the standard tallying process.

In the following, we present two concrete instantiations of this paradigm: MiniMix and HomoRand. These schemes differ in the inputs provided by voters to the pre-tally phase and the way the underlying MPC protocol operates. Before delving into the designs of MiniMix (Section 5.2) and HomoRand (Section 5.3), we first outline their shared structure in Section 5.1, following the general voting system definition in Definition 4.1. We then give the correctness and the security theorem statements of the constructions in Section 6.

5.1 Overview

Key Generation. The election authority EA initiates the setup by running the SetupElection algorithm, generating a public/secret key pair $(\mathsf{PK}, \mathsf{SK})$ (Figure 2 and Figure 3). The public key PK is posted on a public bulletin board PBB, while the private key SK is shared among T talliers using a threshold decryption scheme. Key generation can be distributed securely in prime-order groups through standard techniques.

Voting Phase. To cast a vote v, a voter executes the Vote algorithm. In principle, this procedure begins by secret-sharing v into n shares using a t-out-of-n threshold secret sharing scheme (Section 2). Each share is then independently encrypted under a TREnc (Section 2), producing ballot shares $\{\mathsf{b}_i\}_{i\in[n]}$. The complete ballot b consists of this set of encrypted shares.

Each b_i is then submitted to a distinct randomizing server, while the corresponding trace $\tau = \mathsf{TraceBallot}(\mathsf{b})$ is simultaneously transmitted to PBB. Upon receiving a b_i, a server performs a local verification using the validity checks defined by TREnc, optionally including a zero-knowledge proof. It further ensures that no duplicate traces are present with respect to all traces on PBB. Valid ballot shares are then re-randomized using $\mathsf{ProcessBallot}(\mathsf{b}_i)$ and made available on the bulletin board. Voters can later verify inclusion of their vote by querying VerifyVote on the trace τ and confirming its appearance on PBB.

Tallying Phase. After voting concludes, the tallying phase begins. Talliers aggregate ballot shares by comparing the traces on PBB with the complete traces τ posted by each voter. The Valid function checks whether at least t valid shares are present for each ballot. If the condition is met, the CPA components are extracted from the corresponding shares and combined using Lagrange interpolation. Owing to the linearity of the interpolation, this process can be performed directly in the encrypted domain, yielding a final ciphertext that encrypts the original vote. Ballots failing the Valid check are discarded.

As noted by Doan et al. [8], this reconstruction step is critical for mitigating adversarial influence. A malicious voter may attempt to deviate from the protocol by submitting malformed or inconsistent shares. However, since all valid ballot shares on the bulletin board (possibly more than t) are combined, only one message can be reconstructed. Importantly, the adversary cannot anticipate which subset of shares will be successfully posted (e.g., due to the random failure of non-operational randomizers). Consequently, to ensure that the tally reflects their intended vote, even adversarial voters are incentivized to submit a consistent and correct ballot.

Finally, the reconstructed ciphertexts corresponding to valid ballots are submitted to the $\mathsf{PreTally}$ function, which checks that the encrypted vote is within the valid range (i.e., $\{v_1, v_2, \ldots, v_d\}$). Depending on its inputs, the $\mathsf{PreTally}$ function behaves differently. In the $\mathsf{MiniMix}$ construction (Figure 2), which utilizes the MPC-$\mathsf{MiniMix}$ protocol from Algorithm 1, the voter submits only the encrypted vote. In contrast, the $\mathsf{HomoRand}$ construction (Figure 3) uses the MPC-$\mathsf{HomoRand}$ protocol from Algorithms 2 and 3, where the voter also sends additional values to assist the participating parties in verifying the validity of the encrypted vote. Depending on the deployment scenario, the participating parties may include the talliers or, alternatively, external authorities. For simplicity, we assume that the talliers are responsible for performing this validation in the current setting. Invalid ballots are discarded based on the outcome of the $\mathsf{PreTally}$ check. The talliers then run the Tally function to compute the election result r based on the valid ones according to the election rules. A proof of correctness Π is generated and can be verified by anyone using the $\mathsf{VerifyResult}$ algorithm. In the instantiations below, the number of servers n and the threshold t are implicit inputs for all algorithms.

Relationship between correctness and receipt-freeness thresholds. Our constructions rely on a secret sharing scheme where at least t valid ballot shares are needed to reconstruct a vote. This implies that the system can tolerate up to $n-t$ missing or invalid shares without affecting correctness. On the privacy side, receipt-freeness holds as long as the reconstruction includes at least one honestly rerandomized share. So even if up to $t-1$ ballot shares are maliciously processed, it is infeasible for the adversary to compromise voter privacy. In other words, the system remains secure against up to $t-1$ compromised processing servers, which matches the tight bound shown in prior work by Doan et al. [8].

5.2 The MiniMix Construction

In this scheme, the input to the $\mathsf{PreTally}$ is a randomized ballot $\mathsf{b} = \{\mathsf{b}_i\}_{i\in[n]}$ randomized by randomizers, where each b_i is a TREnc's encryption of a share x_i of the voter's intended message v.

To initiate the $\mathsf{PreTally}$ procedure, the talliers (and optionally the public) verify each ballot share b_i by applying the $\mathsf{TREnc.Ver}$ function (see Section 2). A ballot b is deemed valid by $\mathsf{Valid}(\mathsf{BB}, \mathsf{b})$ (see Figure 2) if at least t valid shares are posted on the public bulletin board. Upon successful validation, the combination algorithm $\mathsf{Combine}$ is publicly executed by any of the talliers. It takes as input the homomorphic components of all available valid ballot shares–up to n shares–and outputs a combined ciphertext $\boldsymbol{C}_0$, representing the encrypted vote v. Due to the properties of Lagrange interpolation, reconstructing with more than t valid shares preserves the correctness of the resulting ciphertext.

Subsequently, T talliers jointly executes the MPC-$\mathsf{MiniMix}$ as described in Algorithm 1 with the input $\boldsymbol{C} = \{\boldsymbol{C}_j\}_{j\in[d]}$, where $\boldsymbol{C}_j = \boldsymbol{C}_0 \cdot \mathsf{CPA}(\mathsf{PK}, -v_j)$. In particular, each tallier T_k for $k \in [T]$ shuffles the d ciphertexts in $\boldsymbol{C}$, applying an independent random factor. It also produces a publicly verifiable ZKPoK proof, attesting to correctness. Misbehaving parties are identified through proof verification. After the last tallier has completed their respective rounds, all the talliers jointly decrypt the final output $\boldsymbol{C}^{(T)} = \{\boldsymbol{C}_j^{(T)}\}_{j\in[d]}$. As $\boldsymbol{C}^{(T)}$ is encrypted under the TREnc's PK, decryption proceeds using $\mathsf{TREnc.Dec}(\mathsf{SK}, \cdot)$ (Algorithm 1, line 22). The $\mathsf{PreTally}$ procedure outputs 1 as soon as one of the decrypted ciphertexts equals 0, confirming that the original vote v belongs to the designated valid range; otherwise, it outputs 0. Invalid votes are discarded, and the Tally function is applied to the combined ciphertext $\boldsymbol{C}_0$ of each valid ballot to compute the final outcome according to the election rules. A formal correctness proof of the MPC-$\mathsf{MiniMix}$ is given in Appendix A.1.

Discussion. It is worth noting that the $\mathsf{MiniMix}$ construction can be adapted to minimize online-phase computation. In the presented variant, the shuffle is performed after a ballot is available on the PBB, which may lead to inefficiencies such as increased latency and synchronization delays.

As an alternative, the authorities may jointly shuffle the encrypted vote domain $\{\boldsymbol{V}_j = \mathsf{CPA}(v_j)\}_{j\in[d]}$ under a secret permutation π, yielding a randomized sequence $\{\boldsymbol{V}'_{\pi(j)}\}_{j\in[d]}$. Given a combined ciphertext $\boldsymbol{C}_0$ (as defined earlier), they compute $\boldsymbol{C}'_j = \boldsymbol{C}_0/\boldsymbol{V}'_{\pi(j)}$ for each $j \in [d]$, resulting in ciphertexts encrypting $(\mathsf{v} - v_{\pi(j)})$. These can be tested for equality to zero to verify whether v belongs to the valid domain. As π remains hidden, the test leaks no information about the actual vote. To avoid linkability, each ballot must be verified against a freshly shuffled domain encryption. Otherwise, repeated shuffles could reveal identical vote positions, enabling correlation across ballots. Hence, the number of domain shuffles must scale with the number of ballots or eligible voters.

5.3 The HomoRand Construction

The input to $\mathsf{PreTally}$ is a randomized ballot $\mathsf{b} = \{\mathsf{b}_i\}_{i\in[n]}$, where each b_i now consists of l components b_{ji} for $j \in [l]$. To this end, the vote v is first decomposed into an l-bit string $\{b_j\}_{j\in[l]}$, and each bit b_j is shared using t-out-of-n secret sharing scheme to obtain n share $\{b_{ji}\}_{i\in[n]}$. Each share b_{ji} is then encrypted separately under two TREnc public keys, yielding a pair of ciphertexts in $\mathbb{G}$ and $\hat{\mathbb{G}}$ (see Vote algorithm, Figure 3, Appendix C). A non-interactive randomizable proof (e.g., Groth-Sahai proofs [10]) accompanies each pair, proving consistency across the two encryptions, i.e., demonstrating that they indeed encrypt the same value. As a result, each ballot share b_i consists of two TREnc ciphertexts and a consistency proof, which are subsequently re-randomized by a randomizer and the resulting share is made available on the public board. Details on the construction are deferred to Appendix C.3.

The $\mathsf{PreTally}$ protocol begins by validating each ballot b posted on the public bulletin board. This involves applying $\mathsf{TREnc.Ver}$ to the ciphertexts and invokes $\mathsf{PrfVerify}$ on the associated proof. A ballot is deemed valid if, for every bit index $j \in [d]$, there exist at least t valid shares $\{\mathsf{b}_{ji}\}_i$ posted on the board. Once a ballot passes verification, the associated proofs are discarded.

SetupElection (λ)

$(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{TREnc.Gen}(1^\lambda)$
return PK

Vote(id, v[, aux])

$\{x_1, \ldots, x_n\} \leftarrow \mathsf{Share}(\mathsf{PK}, t, \mathsf{v})$
if aux is empty **for** $i = 1$ to n do
  $\mathsf{lk}_i \leftarrow\$ \mathsf{TREnc.LGen}(\mathsf{PK})$
else $\{\mathsf{lk}_i\}_{i=1}^n \leftarrow \mathsf{aux}$
for $i = 1$ to n do
  $\mathsf{b}_i \leftarrow \mathsf{TREnc.LEnc}(\mathsf{PK}, \mathsf{lk}_i, x_i)$
return $\mathsf{b} = \{\mathsf{b}_i\}_{i=1}^n$

Valid(BB, b)

if $\exists \mathsf{b}' \in \mathsf{BB} \wedge \exists \tau_i' \subset \mathsf{TraceBallot}(\mathsf{b}')$:
$\tau_i' \subset \mathsf{TraceBallot}(\mathsf{b})$ **then return** $\perp$
$k = 0$
for $i = 1$ to $|\mathsf{b}|$ do
  if $\mathsf{TREnc.Ver}(\mathsf{PK}, \mathsf{b}_i) = 1$
   then $k \leftarrow k + 1$
if $k \geq t$ **then return** 1 **else return** 0

ProcessBallot(b_i)

return $\mathsf{TREnc.Rand}(\mathsf{PK}, \mathsf{b}_i)$

TraceBallot(b)

$\tau_i \leftarrow \mathsf{TREnc.Trace}(\mathsf{b}_i)$
return $\tau = \{\tau_i\}_{i=1}^n$

PreTally (BB, SK, b)

if $\mathsf{Valid}(\mathsf{BB}, \mathsf{b}) = 0$ **then return** 0
for $i = 1$ to n do
  $\boldsymbol{c}_i \leftarrow \mathsf{TREnc.Strip}(\mathsf{PK}, \mathsf{b}_i)$
$\boldsymbol{C}_0 \leftarrow \mathsf{Combine}(n, t, \{\boldsymbol{c}_i\}_{i=1}^{\leq n})$
for $j = 1$ to d do
  $\boldsymbol{C}_j \leftarrow \boldsymbol{C}_0 \cdot \mathsf{CPA}(\mathsf{PK}, -v_j)$
return $\text{MPC-}\mathsf{MiniMix}(\mathsf{PK}, \mathsf{SK}, \boldsymbol{C} = \{\boldsymbol{C}_j\}_{j=1}^d)$

VerifyVote(PBB, τ)

if $\exists \mathsf{b} \in \mathsf{PBB} : \mathsf{Valid}(\mathsf{b}) \wedge \tau == \mathsf{TraceBallot}(\mathsf{b})$
then return 1 **else return** 0

Fig. 2: MiniMix instantiation of our voting scheme.

Next, the Combine algorithm aggregates the homomorphic components of the accepted ballot shares (up to n) to reconstruct two CPA encryptions of $\{b_j\}_{j\in[l]}$ in $\mathbb{G}$ and $\hat{\mathbb{G}}$. The protocol then invokes the MPC-HomoRand procedure from Algorithms 2 and 3 to jointly verify that each b_j is a bit. Construction details are provided in Appendix C.3, with a correctness proof of the MPC-HomoRand given in Appendix A.2.

5.4 Efficiency Discussion

Table 1 compares the computational costs incurred during the casting of a single ballot under the two proposed constructions. All costs reported are per voter, per randomizer, or per tallier, as appropriate.

Computational Costs. On the voter's side, the Vote algorithm in MiniMix requires n invocations of TREnc.Enc, as each ballot share is encrypted independently. In contrast, HomoRand imposes a higher load: it requires $2ln$ encryptions (two per bit per share) and an additional $16ln$ exponentiations to generate Groth-Sahai consistency proofs [10] across all $i \in [n]$ and $j \in [l]$. Detailed technical procedures for computing these proofs can be found in Appendix C.3. Each randomizer, executing ProcessBallot, performs n rerandomizations in MiniMix, while in HomoRand, this increases to $2ln$, alongside rerandomizing the associated proofs. Talliers executing PreTally incur costs primarily from the underlying MPC protocols. In MPC-MiniMix (Algorithm 1), each tallier (i) shuffles d ciphertexts with verifiable proofs at a cost denoted $\mathsf{shuffle}(d)$ (lines 5–10); (ii) applies random factors to shuffled ciphertexts with proofs (lines 12–14), amounting to d rand

	SetupElection	Vote	ProcessBallot	PreTally
MiniMix	$1 \cdot \mathsf{TREnc.Gen}$	$n \cdot \mathsf{TREnc.Enc}$	$n \cdot \mathsf{TREnc.Rand}$	$\mathsf{shuffle}(d) + d \cdot \mathsf{rand} + d \cdot \mathsf{dec}$
HomoRand	$2 \cdot \mathsf{TREnc.Gen}$	$2ln \cdot \mathsf{TREnc.Enc}$ $\mathbb{G}^{16ln} \cdot \hat{\mathbb{G}}^{16ln}$	$2ln \cdot \mathsf{TREnc.Rand}$ $\mathbb{G}^{14ln} \cdot \hat{\mathbb{G}}^{14ln}$	$2l \cdot \mathsf{rand} + (l+1) \cdot \mathsf{dec}$

Table 1: Computational cost per ballot under each construction.

operations; and (iii) participates in the joint decryption of d ciphertexts (lines 21–26), contributing d dec operations. In MPC-HomoRand, each tallier performs $2l$ rand operations (Algorithm 2, lines 2–8) and engages in l+1 dec operations (Algorithm 3, lines 10 and 17).

Verification Costs. Anyone, including external auditors or voters, can verify the PreTally result by checking the posted proofs. In HomoRand, this involves verifying $2l$ rerandomizations and l+1 decryptions. In MiniMix, the verification requires $2l \cdot \mathsf{rand} + (l+1) \cdot \mathsf{dec}$ while in MiniMix it involves $d \cdot (\mathsf{rand} + \mathsf{dec})$, plus $\mathsf{shuffle}(d)$ proofs from each of the T talliers. Since $l = \log_2(d)$, the relative verification cost ratio scales as roughly $d/(2\log_2 d)$ in favor of HomoRand for large domains d, though MiniMix incurs additional overhead from shuffle proofs, which scale with both d and T.

While HomoRand achieves PreTally costs that scale with $l = \log_2(d)$, asymptotically outperforming the linear-in-d cost of MiniMix, this efficiency comes at the price of substantially higher computational overhead for voters and randomizers. Specifically, the cost of Vote in HomoRand scales with both n and l, due to bitwise encryption and the generation of Groth-Sahai proofs. In contrast, MiniMix imposes minimal and domain-independent client-side costs, making it an attractive choice in settings where lightweight voting is essential and the number of valid vote options is modest.

Conversely, HomoRand may be preferable in settings where the verification of PreTally is a critical concern–such as public audits or large-scale elections–particularly when voter-side computation is not a bottleneck and the domain size d is large enough to outweigh its setup and voting costs.

6 Security of the Voting Schemes

The voting schemes presented in the previous section satisfy both *threshold receipt-freeness* and *threshold correctness*, with formal proofs provided in Appendix B.

Theorem 6.1 (Threshold Receipt-Freeness). *Let* TREnc *be a* TCCA*-secure and verifiable encryption scheme, and let the proof systems employed for the pre-tally and for verifying tally correctness be ZKPoK and zero-knowledge, respectively. Then both constructions achieve threshold receipt-freeness under a t-out-of-n sharing scheme with threshold $\mathsf{t_{rf}} = t-1$:*

– *For the* MiniMix *construction:*

$$\Pr[\mathsf{Exp}^{\mathsf{deceive}}_{\mathcal{A},\mathcal{V},\mathsf{t_{rf}}}(\lambda) = 1] \leq \varepsilon_{\mathrm{verif}}, \quad \mathsf{Adv}^{\mathsf{rf},\mathsf{t_{rf}},\beta}_{\mathcal{A},\mathcal{V}}(1^\lambda) \leq \varepsilon_{\mathrm{ZK}} + q(n-\mathsf{t_{rf}})\varepsilon_{\mathrm{tcca}};$$

– *For the* $\mathsf{HomoRand}$ *construction:*
$\Pr[\mathsf{Exp}^{\mathsf{deceive}}_{\mathcal{A},\mathcal{V},\mathsf{t_{rf}}}(\lambda)=1] \le \varepsilon_{\mathrm{verif}}$, $\mathsf{Adv}^{\mathsf{rf},\mathsf{t_{rf}},\beta}_{\mathcal{A},\mathcal{V}}(1^\lambda) \le \varepsilon_{\mathrm{ZK}} + lq(n-\mathsf{t_{rf}})(2\varepsilon_{\mathrm{tcca}} + \varepsilon_{\mathsf{sxdh}})$.

Here, l is the bit-length of the vote domain; $\varepsilon_{\mathrm{ZK}}$, $\varepsilon_{\mathrm{verif}}$, and $\varepsilon_{\mathrm{tcca}}$ bound the adversarial advantage against the ZK proof system, verifiability, and TCCA-*security of* TREnc*, respectively; q is the number of ballot-append queries.*

Theorem 6.2 (Threshold Correctness). *Let* TREnc *be traceable and verifiable, and let the employed proof systems be sound. Then both constructions achieve threshold correctness with $\mathsf{t_{corr}} = t-1$ under a t-out-of-n secret sharing scheme. For any efficient adversary $\mathcal{A}$ making q ballot-append queries:*

– *For the* $\mathsf{MiniMix}$ *construction,* $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda)=1] \le qn\varepsilon_{\mathrm{trace}}$*;*
– *For the* $\mathsf{HomoRand}$ *construction,* $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda)=1] \le 2qln\varepsilon_{\mathrm{trace}}$*,*

where $\varepsilon_{\mathrm{trace}}$ is the where l is the bit-length of the vote domain, and $\varepsilon_{\mathrm{trace}}$ bounds the advantage against traceability of TREnc*.*

Verifiability. The proposed voting system satisfies both *individual* and *universal verifiability*, ensuring that honest voters can confirm the inclusion of their ballots and that the final result reflects all valid inputs.
Individual Verifiability. In the $\mathsf{MiniMix}$ construction, the voting process directly mirrors that of the receipt-free protocol by Doan et al. [8], and thus inherits individual verifiability via the traceability property of TREnc. In the $\mathsf{HomoRand}$ construction, voters additionally submit consistency proofs for all $j \in [l], i \in [n]$, which certify the well-formedness of the ballot shares. These auxiliary proofs do not modify the two ciphertexts, which are generated solely using TREnc. As a result, traceability ensures that no efficient adversary, even one with access to the secret decryption key, can produce a different ciphertext with the same trace that decrypts to a different message.
Universal Verifiability. Both constructions provide universal verifiability via he use of publicly verifiable ZKPoKs in the $\mathsf{PreTally}$ and Tally phases. In the $\mathsf{PreTally}$ phase, these proofs certify that all ballots accepted for tallying encrypt votes within the prescribed domain. The subsequent Tally phase performs a verifiable tallying of the accepted ciphertexts, enabling any observer to check the correctness of the final outcome. Together, they enable any external observer to independently verify the correctness of the final result without compromising ballot privacy.

7 Conclusion

We propose a new direction for validating encrypted ballots in threshold receipt-free voting systems, where ballots are non-interactively rerandomized by multiple independent ballot processing servers. Our approach introduces a pre-tally validation phase executed in a multiparty computation (MPC) style, allowing authorities to verify the validity of encrypted votes without decrypting them or requiring voter-supplied validity proofs.

We develop two constructions that achieve threshold receipt-freeness and verifiability while addressing privacy risks present in prior mixnet-based systems. Both schemes support homomorphic or mixnet-based tallying, depending on vote range constraints, and crucially reveal only the validity status of a ballot, nothing more. To our knowledge, this is the first threshold receipt-free solution to achieve better privacy independently of the tallying technique.

Our efficiency analysis recommends using MiniMix when voter-side efficiency is critical and the number of valid vote options is small or moderate, while HomoRand is better suited for large or complex vote domains. In addition to our core designs, we also discuss how existing validity-check techniques can be adapted to fit our pre-tally framework. An open direction is to reduce the computational and communication complexity of the pre-tally process, or to devise alternative mechanisms such as randomizable validity proofs that can be verified in a single-pass setting, even in the presence of fully malicious randomizers.

References

1. 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 94. pp. 544–553. ACM (1994)
2. Blazy, O., Fuchsbauer, G., Pointcheval, D., Vergnaud, D.: Signatures on randomizable ciphertexts. In: Public Key Cryptography–PKC 2011. LNCS, vol. 6571, pp. 403–422. Springer (2011)
3. 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. pp. 1614–1625. ACM (2016)
4. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Advances in Cryptology - EUROCRYPT '97. pp. 103–118. Springer (1997)
5. Devillez, H., Pereira, O., Peters, T.: Traceable receipt-free encryption. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 273–303. Springer (2022)
6. Devillez, H., Pereira, O., Peters, T., Yang, Q.: Can we cast a ballot as intended and be receipt free? In: IEEE Symposium on Security and Privacy, SP 2024. pp. 3440–3457. IEEE (2024)
7. Doan, T.V.T., Pereira, O., Peters, T.: Encryption mechanisms for receipt-free and perfectly private verifiable elections. In: International Conference on Applied Cryptography and Network Security. pp. 257–287. Springer (2024)
8. Doan, T.V.T., Pereira, O., Peters, T.: Threshold receipt-free single-pass evoting. In: Electronic Voting - 9th International Joint Conference, E-Vote-ID. pp. 20–36. Springer (2024)
9. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for np. In: Advances in Cryptology-EUROCRYPT 2006. pp. 339–358. Springer (2006)
10. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 415–432. Springer (2008)

11. Hirt, M.: Multi party computation: Efficient protocols, general adversaries, and voting (2001)
12. McMurtry, E., Pereira, O., Teague, V.: When is a test not a proof? In: Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security. pp. 23–41. Springer (2020)
13. Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-snarks: Zero-knowledge proofs for distributed secrets. In: 31st USENIX Security Symposium, USENIX Security. pp. 4291–4308. USENIX Association (2022)
14. Shamir, A.: How to share a secret. Communications of the ACM pp. 612–613 (1979)

A MPC protocols' Correctness

A.1 MPC-MiniMix

Theorem A.1. *Assuming all parties act honestly, the MPC-*MiniMix *protocol (Algorithm 1) returns* 1 *if and only if the encrypted message m lies within the designated valid domain $\mathcal{V} = \{v_1, \dots, v_d\}$.*

Proof. The protocol takes as input a vector of ciphertexts $\boldsymbol{C} = \{\boldsymbol{C}_j\}_{j \in [d]}$, where each $\boldsymbol{C}_j = \mathsf{CPA}(\mathsf{PK}, m - v_j)$. Let $P_{k,\dots,T}$ denote the composition of permutations $P_k \circ P_{k+1} \circ \cdots \circ P_T$, and let $P := P_1 \circ \cdots \circ P_T$ denote the global permutation applied to the ciphertexts. Each party T_k (for $k \in [T]$) performs two operations:

1. It applies a secret shuffle P_k to permute the ciphertexts (lines 5–10).
2. It re-randomizes each ciphertext by exponentiating it with fresh non-zero randomness $\alpha_j^{(k)}$ (lines 11–15).

Each party also produces zero-knowledge proofs (ZKPoKs) demonstrating the correctness of the shuffle and the exponentiation. Since all parties are honest, all such proofs (lines 10 and 14) are valid, and verification at line 17 succeeds. Thus, the system proceeds without aborts, and the transformed ciphertexts $\boldsymbol{C}^{(k)}$ are passed to the next party.

After all T rounds, the final ciphertext vector $\boldsymbol{C}^{(T)}$ satisfies:

$$\boldsymbol{C}^{(T)}_{P(j)} = \mathsf{CPA}\left(\mathsf{PK},\ (m - v_j) \cdot \prod_{k=1}^{T} \alpha^{(k)}_{P_{k,\dots,T}(j)}\right),$$

where all $\alpha^{(k)}_{P_{k,\dots,T}(j)} \in \mathbb{Z}_q^*$ due to honest randomness generation. (We omit explicit randomness from the notation for clarity.)

In the final step (lines 21–26), all talliers jointly decrypt each ciphertext using a threshold decryption procedure. Given correctness of the decryption scheme and the honesty of all parties, this step reliably reveals the plaintext of each ciphertext.

If $m \in \mathcal{V}$, then for some $j \in [d]$, we have $m - v_j = 0$, and so $\mathsf{Dec}(\mathsf{SK}, \boldsymbol{C}^{(T)}_{P(j)})$ decrypts to 0, despite any multiplicative randomization. Conversely, if $m \notin \mathcal{V}$, then $m - v_j \neq 0$ for all j, and each $\boldsymbol{C}^{(T)}_{P(j)}$ decrypts to a non-zero value due to the entropy introduced by the non-zero randomizing factors $\alpha^{(k)}_{P_{k,\dots,T}(j)}$. Thus, the protocol outputs 1 if and only if $m \in \mathcal{V}$, as claimed.

A.2 MPC-HomoRand

Theorem A.2. *Assuming all parties act honestly, the MPC-*HomoRand *protocol (Algorithms 2 and 3) outputs* 1 *if and only if the encrypted message m lies within the designated valid domain $\mathcal{V} = \{0, \ldots, 2^l - 1\}$.*

Proof. Let the inputs to Algorithm 2 be $\boldsymbol{C} = \{(\boldsymbol{c}_j, \hat{\boldsymbol{c}}_j)\}_{j\in[l]}$, where each $\boldsymbol{c}_j = \mathsf{CPA}(\mathsf{PK}_1, G^{b_j})$ and $\hat{\boldsymbol{c}}_j = \mathsf{CPA}(\mathsf{PK}_2, \hat{G}^{b_j})$ is an encryption of the j-th bit of the underlying message m under independent public keys PK_1 and PK_2 respectively.

For concreteness, we instantiate CPA with a structure-preserving, additive homomorphic encryption scheme defined over prime-order groups. Specifically, an encryption under $\mathsf{PK}_1 = (G, H, f) \in \mathbb{G}^3$, with secret key $\mathsf{SK}_1 = (\eta_1, \epsilon_1)$ and $f = G^{\eta_1} H^{\epsilon_1}$, takes the form:

$$\boldsymbol{c}_j = (G^{b_j} f^{\alpha_j}, G^{\alpha_j}, H^{\alpha_j}) \quad \text{for } \alpha_j \leftarrow\$\, \mathbb{Z}_p.$$

An analogous form holds for $\hat{\boldsymbol{c}}_j$ under $\mathsf{PK}_2 = (\hat{G}, \hat{H}, \hat{f})$ with randomness β_j.

Each party T_k (for $k \in [T]$) independently rerandomizes $\boldsymbol{c}_j$ and $\hat{\boldsymbol{c}}_j$ (lines 4–5) for random exponents $\gamma_{k,j}, \lambda_{k,j}, r_{k,j}, s_{k,j} \leftarrow\$\, \mathbb{Z}_p \setminus \{0\}$. Explicitly, we have:

$$\boldsymbol{c}''_{k,j} = (G^{b_j\gamma_{k,j}+\lambda_{k,j}}, 1, 1) \cdot (f^{\gamma_{k,j}\alpha_j + r_{k,j}}, g^{\gamma_{k,j}\alpha_j + r_{k,j}}, h^{\gamma_{k,j}\alpha_j + r_{k,j}}),$$
$$\hat{\boldsymbol{c}}''_{k,j} = (\hat{G}^{(1-b_j)\lambda_{k,j}}, 1, 1) \cdot (\hat{f}^{s_{k,j} - \beta_j\lambda_{k,j}}, \hat{g}^{s_{k,j} - \beta_j\lambda_{k,j}}, \hat{h}^{s_{k,j} - \beta_j\lambda_{k,j}}).$$

Each randomized ciphertext is accompanied by a ZKPoK $\pi_{k,j}$ attesting to the correctness of the transformation, and due to the honesty of all parties, all proofs are valid and accepted in Algorithm 3 (line 4).

The next step involves publicly aggregating the rerandomized ciphertexts (lines 8–9), which can be done by any party of them and verified by any party:

$$\boldsymbol{c}''_j = \prod_{k\in[T]} \boldsymbol{c}''_{k,j} = \left(G^{b_j\gamma_j+\lambda_j}, 1, 1\right) \cdot (f^{\alpha_j\gamma_j + r_j}, g^{\alpha_j\gamma_j + r_j}, h^{\alpha_j\gamma_j + r_j}),$$
$$\hat{\boldsymbol{c}}''_j = \prod_{k\in[T]} \hat{\boldsymbol{c}}''_{k,j} = \left(\hat{G}^{(1-b_j)\lambda_j}, 1, 1\right) \cdot (\hat{f}^{s_j - \beta_j\lambda_j}, \hat{g}^{s_j - \beta_j\lambda_j}, \hat{h}^{s_j - \beta_j\lambda_j}),$$

where $\gamma_j = \sum_k \gamma_{k,j}$, $\lambda_j = \sum_k \lambda_{k,j}$, and similarly for r_j, s_j.

Decryption of $\boldsymbol{c}''_j$ (line 10) yields:

$$X_j = G^{b_j\gamma_j+\lambda_j},$$

since all parties are honest and the decryption is correct.

Next, the parties compute:

$$\bar{\boldsymbol{c}}'_j := \mathsf{CPA}(\mathsf{PK}_2, \hat{G}; 0)/\hat{\boldsymbol{c}}_j,$$

and evaluate the pairings:

$$a_j = \frac{e(X_j, \bar{c}'_{0j})}{e(G, \hat{c}''_{0j})} = e(G, \hat{G})^{b_j(1-b_j)\gamma_j} \cdot e(G, \hat{f})^{-\beta_j b_j \gamma_j - s_j},$$
$$b_j = \frac{e(X_j, \bar{c}'_{1j})}{e(G, \hat{c}''_{1j})} = e(G, \hat{g})^{-\beta_j b_j \gamma_j - s_j},$$
$$c_j = \frac{e(X_j, \bar{c}'_{2j})}{e(G, \hat{c}''_{2j})} = e(G, \hat{h})^{-\beta_j b_j \gamma_j - s_j}.$$

In the final step (line 17), the parties jointly decrypt:

$$Y := a \cdot b^{-\eta_2} \cdot c^{-\epsilon_2} = \prod_{j \in [l]} e(G, \hat{G})^{b_j(1-b_j)\gamma_j},$$

where the exponent simplifies due to the key relations: $\hat{f} = \hat{G}^{\eta_2}\hat{H}^{\epsilon_2}$.

Observe that $b_j(1-b_j) = 0$ if and only if $b_j \in \{0,1\}$, and $\gamma_j \neq 0$ for all j by the soundness of the zero-knowledge proofs. Hence, $Y = 1_{\mathbb{G}_\mathcal{T}}$ if and only if all b_j are bits. In that case, $m = \sum_j b_j 2^j \in \mathcal{V}$, and the protocol returns 1.

Conversely, if $m \notin \mathcal{V}$, then there exists some j^* such that $b_{j^*} \notin \{0,1\}$, leading to $b_{j^*}(1-b_{j^*})\gamma_{j^*} \neq 0$ and thus $Y \neq 1_{\mathbb{G}_\mathcal{T}}$. In this case, the protocol returns 0, as desired.

B Security of the voting schemes

In this section, we prove that our voting schemes described in the previous section is threshold correct (Section B.1) and threshold receipt-free (Section B.2).

B.1 Threshold Correctness

Threshold correctness is captured by the experiment $\mathsf{Exp}^{\mathsf{corr},\mathsf{t}_\mathsf{corr}}_{\mathcal{A},\mathcal{V}}(\lambda)$, which models the integrity of election outcomes in the presence of adversarially modified ballots. The experiment maintains two internally consistent election views, each initialized with a set of honestly generated ballots and corresponding tracing information. The adversary $\mathcal{A}$ may attempt to manipulate these ballots by reconstructing new valid ones (via $\mathcal{O}\mathsf{append}$) that omit up to t_corr shares and submitting them into both views. The adversary can also introduce arbitrary valid ballots of its own (via $\mathcal{O}\mathsf{cast}$) and query the public bulletin board and tally results for each view. The experiment outputs 1 if the two election views yield different final results–indicating a breach of threshold correctness. A secure system ensures that such an event occurs only with negligible probability.

Theorem B.1. *Let* TREnc *be traceable and verifiable, and let the employed proof systems be sound. Then both constructions achieve threshold correctness with* $\mathsf{t}_\mathsf{corr} = t-1$ *under a* t*-out-of-*n *secret sharing scheme. For any efficient adversary* $\mathcal{A}$ *making* q *ballot-append queries:*

- *For the* $\mathsf{MiniMix}$ *construction,* $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t}_\mathsf{corr}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1] \leq qn\varepsilon_{\mathrm{trace}}$;
- *For the* $\mathsf{HomoRand}$ *construction,* $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t}_\mathsf{corr}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1] \leq 2qln\varepsilon_{\mathrm{trace}}$*,*

where $\varepsilon_{\mathrm{trace}}$ *is the where* l *is the bit-length of the vote domain, and* $\varepsilon_{\mathrm{trace}}$ *bounds the advantage against traceability of* TREnc*.*

Proof. In the experiment $\mathsf{Exp}^{\mathsf{corr},\mathsf{t}_\mathsf{corr}}_{\mathcal{A},\mathcal{V}}(\lambda)$, tthe adversary interacts with two election views by issuing the following types of queries:

$\mathcal{O}\mathsf{cast}$: This oracle appends the same honestly generated, yet potentially malicious, valid ballot to both bulletin boards BB_0 and BB_1. Since both BB_0 and BB_1 are updated identically in this process, this query does not aid the adversary in winning the game under either construction.

$\mathcal{O}$append: Upon this query, two distinct valid ballots, b_0 and b_1, derived in a malicious yet valid manner from an honestly generated ballot b, are appended to BB_0 and BB_1, respectively. These ballots are required to be traceable to b while omitting at most $\mathsf{t_{corr}}$ shares. We analyze both constructions:

- MiniMix **construction:** Each vote share is encrypted under TREnc and labeled with a trace value. Given traceability, any valid share reusing the same trace must encrypt the same plaintext. Thus, the adversary cannot produce two valid ballots with the same trace that decrypt to different vote shares, except with probability at most $\varepsilon_{\text{trace}}$ per share. Since each ballot has n shares and at most q such append operations are allowed, the total advantage is bounded by $qn\varepsilon_{\text{trace}}$.
- HomoRand **construction:** Each vote share b_{ji} (for $j \in [l], i \in [n]$) includes two TREnc ciphertexts along with a proof of consistency (as detailed in Figure 3 and Section C). By traceability, the adversary cannot alter the encrypted messages in the two ciphertext. The adversary can at most break traceability on either ciphertext, giving a per-share failure bound of $2\varepsilon_{\text{trace}}$. As for the consistency proof, while the adversary may attempt arbitrary modifications, these do not influence the underlying encrypted values in the two ciphertexts, and thus do not affect the soundness of the game. Since there are ln such shares per ballot and q queries, the total advantage is bounded by $2qln\varepsilon_{\text{trace}}$.

In both constructions, the aggregation of vote shares via the Combine function operates on inputs provided by PreTally, which are guaranteed to be honest due to the traceability of TREnc and the correctness of the underlying secret sharing scheme. Specifically, any subset of at least $n - \mathsf{t_{corr}}$ shares suffices to reconstruct the original vote. Moreover, by the correctness of the MPC-MiniMix and MPC-HomoRand procedures (established in Section A), the PreTally function deterministically outputs 1 if the reconstructed vote lies within the valid range, and 0 otherwise. Since both views include valid reconstructions and run the same tallying logic, the outputs are necessarily identical unless the adversary has introduced inconsistency via traceability failure.

As a consequence, the adversary's advantage in distinguishing the games is bounded by the traceability error. Specifically, we obtain $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1] \leq qn\varepsilon_{\text{trace}}$ for the MiniMix construction and $\Pr[\mathsf{Exp}^{\mathsf{corr},\mathsf{t_{corr}}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1] \leq 2qln\varepsilon_{\text{trace}}$ for the HomoRand construction by a standard guessing technique.

B.2 Threshold Receipt-Freeness

Threshold RF is defined via two experiments: $\mathsf{Exp}^{\mathsf{deceive}}_{\mathcal{A},\mathcal{V},\mathsf{t_{rf}}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$.

The $\mathsf{Exp}^{\mathsf{deceive}}_{\mathcal{A},\mathcal{V},\mathsf{t_{rf}}}(\lambda)$ formalizes the notion that even under coercion or external pressure to vote for a specific candidate v_0, a voter can still successfully cast their intended vote v_1 without detection, despite partial system compromise. The adversary $\mathcal{A}$, given the public key and control over up to $\mathsf{t_{rf}}$ ballot processing servers, selects two votes v_0 and v_1, as well as a random coins, and attempts to construct a convincing receipt that proves the ballot encodes v_0. However, $\mathcal{A}$ observes only the ballot shares corresponding to the corrupted servers and the public tracking information. The goal is to use the $\mathsf{Deceive}$ algorithm to compute the remaining ballot shares such that the resulting ballot is valid for v_1 but remains indistinguishable from a ballot for v_0 with respect to the adversary's view.

The $\mathsf{Exp}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$ captures the intuition that even if a voter controls up to t_{rf} ballot processing servers, they should be unable to construct a ballot–possibly sampled from an arbitrary distribution–that can serve as a convincing receipt. The experiment runs with a hidden bit $\beta \in \{0,1\}$. When $\beta = 0$, the system operates honestly; when $\beta = 1$, certain components, such as the tally and its associated proof, are simulated using trapdoor information. The adversary is allowed to submit two valid ballots that differ only in the ballot shares processed by honest randomizers, along with known ballot shares corresponding to the compromised t_{rf} servers. These inputs are used to populate two internal ballot boxes, BB_0 and BB_1, which are updated in parallel. However, the adversary only observes the public bulletin board corresponding to BB_β. At any point, it may inspect this view, and eventually it queries the tally oracle to obtain the election outcome and a proof of its correctness, which is simulated if $\beta = 1$. The adversary's task is to guess the bit β. Security holds if no efficient adversary can distinguish between the real and simulated executions with advantage significantly better than random guessing.

Theorem B.2. *Let* TREnc *be a* TCCA*-secure and verifiable encryption scheme, and let the proof systems employed for the pre-tally and for verifying tally correctness be ZKPoK and zero-knowledge, respectively. Then the* $\mathsf{MiniMix}$ *construction achieves threshold receipt-freeness under a t-out-of-n sharing scheme with threshold* $\mathsf{t}_{\mathsf{rf}} = t-1$*. More precisely,*

$$\Pr[\mathsf{Exp}^{\mathsf{deceive}}_{\mathcal{A},\mathcal{V},\mathsf{t}_{\mathsf{rf}}}(\lambda) = 1] \leq \varepsilon_{\mathrm{verif}}, \quad \mathsf{Adv}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},\beta}_{\mathcal{A},\mathcal{V}}(1^\lambda) \leq \varepsilon_{\mathrm{ZK}} + q(n-\mathsf{t}_{\mathsf{rf}})\varepsilon_{\mathrm{tcca}}.$$

Here, l is the bit-length of the vote domain; $\varepsilon_{\mathrm{ZK}}$, $\varepsilon_{\mathrm{verif}}$, and $\varepsilon_{\mathrm{tcca}}$ bound the adversarial advantage against the ZK proof system, verifiability, and TCCA*-security of* TREnc*, respectively; q is the number of ballot-append queries.*

Proof. **The experiment** $\mathsf{Exp}^{\mathsf{deceive},\mathsf{t}_{\mathsf{rf}}}_{\mathcal{A},\mathcal{V}}(\lambda)$**.** Security follows directly from traceability and verifiability of TREnc, and the correctness of the secret sharing scheme as shown in [8].

The experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$**.** In this experiment, an attacker must produce two valid ballots, namely $\mathsf{CT}_0 = \mathsf{B}_0||\mathsf{B}_2$ and $\mathsf{CT}_1 = \mathsf{B}_1||\mathsf{B}_2$, corresponding to the encrypted votes v_0 and v_1. The oracle then verifies that both ballots have identical traces. More precisely, the conditions $\mathsf{Valid}(\mathsf{CT}_0) = \mathsf{Valid}(\mathsf{CT}_1) = 1$ and $\mathsf{TraceBallot}(\mathsf{CT}_0) = \mathsf{TraceBallot}(\mathsf{CT}_1)$ must be met before processing them.

The proof involves a series of indistinguishable games, starting with the experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},0}_{\mathcal{A},\mathcal{V}}(\lambda)$ ($\beta = 0$) and ending with $\mathsf{Exp}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},1}_{\mathcal{A},\mathcal{V}}(\lambda)$ ($\beta = 1$).

$\mathsf{Game}_1(\lambda)$**:** This is the experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},0}_{\mathcal{A},\mathcal{V}}(\lambda)$ with $\beta = 0$, where the honest Tally is called to compute the election result $\mathcal{R}$. By definition, $\Pr[S_1] = \Pr[\mathsf{Exp}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},0}_{\mathcal{A},\mathcal{V}}(\lambda) = 1]$.

$\mathsf{Game}_2(\lambda)$**:** This game proceeds as in Game 1, except that instead of computing the result of BB_0 using the honest Tally, we use the TREnc.Dec oracle to decrypt all the ballot shares in BB_0 and BB_1.

By the correctness of TREnc, the computed election result $\mathcal{R}$ remains identical to the one obtained using the honest Tally. Since the view in Game 2 is indistinguishable to that in Game 1, it follows that $\Pr[S_1] = \Pr[S_2]$.

$\mathsf{Game}_3(\lambda)$**:** This game builds upon Game 2 but introduces a modification in the key generation process. Specifically, the election keys are generated using SimSetupElection, which, in addition to producing the secret key SK, generates an auxiliary trapdoor key to facilitate the simulation of proofs for decryption.

To formalize this, we introduce a simulator $\mathcal{S}$, which receives the keys of all honest talliers as part of its input. For corrupt trustees, $\mathcal{S}$ instantiates a new instance of the adversary for each such tallier and exploits the proof of knowledge (PoK) property of the election administrator's secret key proof to extract the corresponding key. Given that the proof system employed in the voting scheme is zero-knowledge, it follows that the distinguishing advantage between the two games is bounded by $|\Pr[S_2] - \Pr[S_3]| \leq \varepsilon_{\mathrm{ZK}}$.

$\mathsf{Game}_4(\lambda)$**:** In this game, we modify the response behavior to adversarial queries to the $\mathcal{O}$receiptLR oracle by progressively replacing processed ballots from B_0 with those from B_1 within BB_0. The game proceeds as follows.

$\mathsf{Game}_{4,1}(\lambda)$**:** First, in order to compute B'_0, instead of re-randomizing $\mathsf{CT}_{0,1}$, we re-randomize $\mathsf{CT}_{1,1}$, i.e., $\mathsf{B}'_0 = (\mathsf{CT}'_{1,1}, \mathsf{CT}'_{0,2}, \ldots, \mathsf{CT}'_{0,\mathsf{n}-\mathsf{t}_{\mathsf{rf}}})$. The adversary $\mathcal{A}$ can distinguish this modification only if it can determine whether the first element of B'_0 originates from the randomization of $\mathsf{CT}_{0,1}$ or $\mathsf{CT}_{1,1}$. Since each ballot share is a valid TREnc ciphertext, the probability of successfully distinguishing this change is bounded by $\varepsilon_{\mathrm{tcca}}$. The subsequent behavior of the system remains unchanged as the PreTally procedure in Figure 2, we compute $\boldsymbol{C}_0 = \mathsf{Combine}(\mathsf{PK}, \mathsf{B}'_0||\mathsf{B}_2)$, $\boldsymbol{C}_j = \boldsymbol{C}_0 \cdot \mathsf{CPA}(\mathsf{PK}, -v_j)$ for $j \in [d]$, followed by execution of the MiniMix protocol (see Algorithm 1) :

1. The first party T_1 shuffles the ciphertexts $\{\boldsymbol{C}_j\}_{j\in[d]}$ (from line 5 to 12) forwarding the result to the next party, and so on. After all talliers have shuffled, the final party T_T outputs $\boldsymbol{C}^{(T)}$ and $\boldsymbol{\pi}^{(T)}$.
2. All T parties jointly decrypt all ciphertexts in $\boldsymbol{C}^{(T)}$ to verify the vote validity (lines 19-24). The MiniMix algorithm output 1 right after any any ciphertext decrypts to 0. Since the simulator $\mathcal{S}$ knows the plaintexts on BB_0 via TREnc.Dec, it simulates this

decryption process depending on whether the underlying vote is valid or not.

– If the original vote is valid:
  - It randomly selects an index $j^* \in [d]$ and chooses an honest party $\mathcal{K}'$.
  - For each party $\mathcal{K} \neq \mathcal{K}'$, $\mathcal{S}$ computes a valid decryption share $\mathcal{D}_{\mathcal{K}}$ for the ciphertext $C_{j^*}^{(T)} \in \boldsymbol{C}^{(T)}$ using their secret keys.
  - Then, it sets the decryption share $\mathcal{D}_{\mathcal{K}'}$ such that the joint decryption yields 0, i.e., $\mathsf{TREnc.Dec}(\mathsf{SK}, C_{j^*}^{(T)}) = 0$.
  - Finally, $\mathcal{S}$ leverages the zero-knowledge property of the decryption proof system to generate a simulated proof of correctness for $\mathcal{D}_{\mathcal{K}'}$ and posts it.

– If the vote is invalid, then ideally $\mathsf{TREnc.Dec}(\mathsf{SK}, \boldsymbol{C}_j^{(T)}) \neq 0$ for all $j \in [d]$. To this end, the simulator $\mathcal{S}$ can proceed by sampling non-zero values $\{r_j\}_{j\in[d]} \leftarrow\!\!\$\ \mathbb{Z}_p \setminus \{0\}$ and applying a similar simulation strategy as above, ensuring that $\mathsf{TREnc.Dec}(\mathsf{SK}, \boldsymbol{C}_j^{(T)}) = r_j$ for all j.

Consequently, we obtain $|\Pr[S_{4,1}] - \Pr[S_3]| \leq \varepsilon_{\text{tcca}}$.

$\mathsf{Game_{4,i}}(\lambda)$**:** By repeating this process iteratively, each element of B'_0 is replaced with its corresponding element from B'_1. Thus, we derive $|\Pr[S_{4,i-1}] - \Pr[S_{4,i}]| \leq \varepsilon_{\text{tcca}}$.

At the conclusion of Game 4, B'_0 is indistinguishable from B'_1. Consequently, for the first query to the $\mathcal{O}\mathsf{receiptLR}$ oracle, we obtain

$$|\Pr[S_4] - \Pr[S_3]| \leq (n - \mathsf{t_{rf}})\varepsilon_{\text{tcca}}.$$

Applying a hybrid argument over all q queries made by the adversary, we have

$$|\Pr[S_4] - \Pr[S_3]| \leq q(n - \mathsf{t_{rf}})\varepsilon_{\text{tcca}}.$$

As the adversary's view in Game 4 is identical to its view in $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{rf},\mathsf{t_{rf}},1}(\lambda)$, we conclude

$$\mathsf{Adv}_{\mathcal{A},\mathcal{V}}^{\mathsf{rf},\mathsf{t_{rf}},\beta}(1^\lambda) \leq \varepsilon_{\text{ZK}} + q(n - \mathsf{t_{rf}})\varepsilon_{\text{tcca}}.$$

Theorem B.3. *Let* TREnc *be a* TCCA*-secure and verifiable encryption scheme, and let the proof systems employed for the pre-tally and for verifying tally correctness be ZKPoK and zero-knowledge, respectively. Then the* $\mathsf{HomoRand}$ *construction achieves threshold receipt-freeness under a t-out-of-n sharing scheme with threshold* $\mathsf{t_{rf}} = t-1$*. More precisely,*

$$\Pr[\mathsf{Exp}_{\mathcal{A},\mathcal{V},\mathsf{t_{rf}}}^{\mathsf{deceive}}(\lambda) = 1] \leq \varepsilon_{\text{verif}}, \quad \mathsf{Adv}_{\mathcal{A},\mathcal{V}}^{\mathsf{rf},\mathsf{t_{rf}},\beta}(1^\lambda) \leq \varepsilon_{\text{ZK}} + lq(n - \mathsf{t_{rf}})(2\varepsilon_{\text{tcca}} + \varepsilon_{\mathsf{sxdh}}).$$

Here, l is the bit-length of the vote domain; ε_{ZK}, $\varepsilon_{\text{verif}}$, and $\varepsilon_{\text{tcca}}$ bound the adversarial advantage against the ZK proof system, verifiability, and TCCA*-security of* TREnc*, respectively; q is the number of ballot-append queries.*

Proof. **The experiment** $\mathsf{Exp}_{\mathcal{A},\mathcal{V}}^{\mathsf{deceive},\mathsf{t_{rf}}}(\lambda)$**.** Security follows directly from traceability and verifiability of TREnc, and the correctness of the secret sharing scheme as shown in [8].

The experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$**.** The proof involves a series of indistinguishable games, starting with the experiment $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},0}_{\mathcal{A},\mathcal{V}}(\lambda)$ ($\beta = 0$) and ending with $\mathsf{Exp}^{\mathsf{rf},\mathsf{t_{rf}},1}_{\mathcal{A},\mathcal{V}}(\lambda)$ ($\beta = 1$). $\mathsf{Game}_1(\lambda)$, $\mathsf{Game}_2(\lambda)$, and $\mathsf{Game}_3(\lambda)$ proceed as the case of MiniMix above.

$\mathsf{Game}_4(\lambda)$**:** In this game, we modify the response mechanism to adversarial queries to the $\mathcal{O}\mathsf{receiptLR}$ oracle by progressively replacing processed ballots from B_0 with those from B_1 within BB_0. The modifications are introduced in stages as follows:

$\mathsf{Game}_{4,1}(\lambda)$**:** First, in order to compute B'_0, instead of re-randomizing $\mathsf{CT}_{0,1}$, we re-randomize $\mathsf{CT}_{1,1}$, i.e., $\mathsf{B}'_0 = (\mathsf{CT}'_{1,1}, \mathsf{CT}'_{0,2}, \ldots, \mathsf{CT}'_{0,\mathsf{n}-\mathsf{t_{rf}}})$. The adversary $\mathcal{A}$ can distinguish this modification only if it can determine whether the first element of B'_0 originates from the randomization of $\mathsf{CT}_{0,1}$ or $\mathsf{CT}_{1,1}$. Since each ballot share consists of two valid TREnc ciphertexts, accompanied by Groth-Sahai proofs that are witness-indistinguishable with a perfectly hiding common reference string $(\mathbf{crs}_1, \mathbf{crs}_2)$ under SXDH assumption, the probability of successfully distinguishing this change is bounded by $2\varepsilon_{\mathrm{tcca}} + \varepsilon_{\mathsf{sxdh}}$. Following the PreTally procedure in Figure 3, we denote $\boldsymbol{C} = \mathsf{Combine}(\mathsf{PK}, \mathsf{B}'_0 || \mathsf{B}_2)$ and proceed the HomoRand algorithm (see Algorithms 2 and 3) as follows:

1. Each party independently and in parallel executes the steps 3 to 9 on input $\boldsymbol{C}$.
2. Then, all parties collaboratively verify whether the encrypted vote is in the valid range by computing and decryption (a, b, c) (lines 25–31). Since the simulator $\mathcal{S}$ has access to the plaintexts on BB_0 via TREnc.Dec, there are two cases:
   - If the original vote is valid, $\mathcal{S}$ simulates the decryption of (a, b, c) to be 0. More precisely,
     - $\mathcal{S}$ picks one honest party $\mathcal{K}'$, computes the correct decryption shares $\mathcal{D}_{\mathcal{K}}$ for all other parties $\mathcal{K} \neq \mathcal{K}'$ based on b, c (as $\mathcal{S}$ has their secret keys).
     - It sets the decryption share of $\mathcal{K}'$ as: $\mathcal{D}'_{\mathcal{K}} = a / \prod_{\mathcal{K} \neq \mathcal{K}'} \mathcal{D}_{\mathcal{K}}$.
     - $\mathcal{S}$ generates a simulated proof of correctness for $\mathcal{D}_{\mathcal{K}'}$.
   - If the vote is invalid, then ideally $\mathsf{TREnc.Dec}(\mathsf{SK}_2, (a, b, c)) \neq 1_{G_{\mathcal{T}}}$ for all $j \in [d]$. To this end, $\mathcal{S}$ can proceed by sampling nonzero values $r \leftarrow\$ \mathbb{Z}_p \setminus \{0\}$ and applying a similar simulation strategy as above, ensuring that $\mathsf{TREnc.Dec}(\mathsf{SK}_2, (a, b, c)) = e(G, \hat{G})^r \neq 1_{G_{\mathcal{T}}}$.

Consequently, we obtain $|\Pr[S_{4,1}] - \Pr[S_3]| \leq 2\varepsilon_{\mathrm{tcca}} + \varepsilon_{\mathsf{sxdh}}$.

$\mathsf{Game}_{4,\mathsf{i}}(\lambda)$**:** By repeating this process iteratively, each element of B'_0 is replaced with its corresponding element from B'_1. Thus, we derive $|\Pr[S_{4,i-1}] - \Pr[S_{4,i}]| \leq 2\varepsilon_{\mathrm{tcca}}$.

At the conclusion of Game 4, B'_0 is indistinguishable from B'_1. Consequently, for the first query to the $\mathcal{O}\mathsf{receiptLR}$ oracle, we obtain

$$|\Pr[S_4] - \Pr[S_3]| \leq l(n - \mathsf{t_{rf}})(2\varepsilon_{\mathrm{tcca}} + \varepsilon_{\mathsf{sxdh}}).$$

Applying a hybrid argument over all q queries made by the adversary, we have

$$|\Pr[S_4] - \Pr[S_3]| \leq lq(n - \mathsf{t}_{\mathsf{rf}})(2\varepsilon_{\text{tcca}} + \varepsilon_{\mathsf{sxdh}}).$$

As the adversary's view in Game 4 is identical to its view in $\mathsf{Exp}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},1}_{\mathcal{A},\mathcal{V}}(\lambda)$, we conclude

$$\mathsf{Adv}^{\mathsf{rf},\mathsf{t}_{\mathsf{rf}},\beta}_{\mathcal{A},\mathcal{V}}(1^\lambda) \leq \varepsilon_{\text{ZK}} + lq(n - \mathsf{t}_{\mathsf{rf}})(2\varepsilon_{\text{tcca}} + \varepsilon_{\mathsf{sxdh}}).$$

C Details of the **HomoRand** Construction

C.1 Computational Setting

We rely on an efficient Setup algorithm to generate common public parameters pp. Given a security parameter λ, $\mathsf{Setup}(1^\lambda)$ outputs $\mathsf{pp} = (\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, p, e, g, \hat{g})$ where $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$ are groups of prime order $p > 2^{\text{poly}(\lambda)}$ for some polynomial poly, with $g \leftarrow \mathbb{G}$ and $\hat{g} \leftarrow \hat{\mathbb{G}}$ as generators, and $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ as a bilinear map. In this setting, we assume the SXDH assumption, which states that the DDH problem remains computationally hard in both $\mathbb{G}$ and $\hat{\mathbb{G}}$.

C.2 Groth-Sahai Proofs

The Groth-Sahai (GS) proof system [10] provides efficient non-interactive proofs for satisfiability of pairing-product equations. We use them to prove consistency and knowledge of ciphertexts, which supports perfectly rerandomizable proofs and can be efficiently instantiated in pairing-friendly groups under the SXDH assumption. The GS proof system consists of the following PPT algorithms:

- $\mathsf{GSSetup}(1^\lambda)$: On input a security parameter λ, outputs a common reference string (CRS) $\mathsf{crs} \in \mathbb{G}_1^4 \times \mathbb{G}_2^4$ for which the commitment scheme is perfectly hiding (or perfectly binding, depending on the mode).
- $\mathsf{GSProve}(\mathsf{crs}, x; w)$: On input a CRS crs, a statement x (consisting of a system of pairing-product equations), and a witness w satisfying x, outputs a non-interactive proof π under crs.
- $\mathsf{GSVerify}(\mathsf{crs}, x, \pi)$: On input a CRS crs, a statement x, and a proof π, returns accept if π is a valid proof that x is satisfiable, and reject otherwise.
- $\mathsf{GSRand}(\mathsf{crs}, x, \pi)$: On input a CRS crs, a statement x, and a valid proof π for x, outputs a rerandomized proof π' such that:
  - $\mathsf{GSVerify}(\mathsf{crs}, x, \pi') = \mathsf{accept}$, and
  - π' is computationally indistinguishable from a fresh proof generated by $\mathsf{GSProve}(\mathsf{crs}, x; w)$, assuming crs is generated in the perfectly hiding mode.

Following the standard GS notation, we define the map $\iota : \mathbb{G} \to \mathbb{G}^2$ that maps $X \in \mathbb{G}$ to $\iota(X) = (X, 1)$ and the map $\iota_T : \mathbb{G}_T \to \mathbb{G}_T^2$ that maps $T \in \mathbb{G}_T$ to $\iota_T(T) = (T, 1)$. We also extend the pairing as $E_1 : \mathbb{G}^2 \times \hat{\mathbb{G}} \to$

$\mathbb{G}_T^2$ such that $E_1(\boldsymbol{a}, b) = (e(a_1, b), e(a_2, b))$, $E_2 : \mathbb{G} \times \hat{\mathbb{G}}^2 \to \mathbb{G}_T^2$ such that $E_1(a, \boldsymbol{b}) = (e(a_1, b), e(a_2, b))$, and $E : \mathbb{G}^2 \times \hat{\mathbb{G}}^2 \to \mathbb{G}_T^4$ such that $E(\boldsymbol{a}, \boldsymbol{b}) = (e(a_1, b_1), e(a_2, b_1), e(a_1, b_2), e(a_2, b_2))$, where $\boldsymbol{a} = (a_1, a_2)$ and $\boldsymbol{b} = (b_1, b_2)$. We use the multiplicative notation for vector space operations.

C.3 The HomoRand Construction

We now detail the construction of the $\mathsf{HomoRand}$ voting scheme. As it follows the general framework outlined in Section 5.1, we focus here on the technical specification of the scheme's core functions as defined in Figure 3. We assume that it is straightforward to extract specific parts of the TREnc's public key PK. In particular:

$\mathsf{Strip}(\mathsf{pk})$: Extracts the public parameters to compute the CPA part. For example, in TREnc [5], we have $\mathsf{Strip}(\mathsf{pk}) = (f, g, h)$, where $f = g^\alpha h^\beta$ for secret key (α, β).

$\underline{\mathsf{SetupElection}(\lambda)}$**:** Chooses bilinear groups $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_\mathcal{T})$ of prime order p ($p > 2^{\text{poly}(\lambda)}$) together with $G \leftarrow\$ \mathbb{G}, \hat{G} \leftarrow\$ \hat{\mathbb{G}}$, and

1. Run $\mathsf{TREnc.Gen}(1^\lambda)$ to generate two the secret/public key pairs $(\mathsf{SK}_1, \mathsf{PK}_1)$ and $(\mathsf{SK}_2, \mathsf{PK}_2)$ to encrypt messages in $\mathbb{G}$ and $\hat{\mathbb{G}}$ respectively.
2. Run $\mathsf{GSSetup}(1^\lambda)$ to generate two tuples of 4 random group elements $(\mathbf{crs}_1, \mathbf{crs}_2) \leftarrow\$ \mathbb{G}^4 \times \hat{\mathbb{G}}^4$ such that $\mathbf{crs}_1 = (\boldsymbol{u}_1, \boldsymbol{u}_2)$ is seen as a Groth-Sahai CRS to commit to group elements over $\mathbb{G}$ and $\mathbf{crs}_1 = (\boldsymbol{\varphi}, \boldsymbol{\psi})$ is seen as a Groth-Sahai CRS to commit to group elements over $\hat{\mathbb{G}}$.

The private key consists of $\mathsf{SK} = \{\mathsf{SK}_1, \mathsf{SK}_2\}$ and the public key $\mathsf{PK} = \{G, \hat{G}, \mathbf{crs}_1, \mathbf{crs}_2, \mathsf{PK}_1, \mathsf{PK}_2\}$.

$\underline{\mathsf{Vote}(\mathsf{id}, \mathsf{v}, l, \mathsf{aux})}$ To encrypt a vote v, a voter presents it as a l-bit string $b_1 b_2 \ldots b_l$, then conducts the following steps of $\mathsf{Enc}(\mathsf{PK}, b, \mathsf{aux})$ to encrypt <u>each bit</u> $b \in \{b_1, b_2, \ldots, b_l\}$:

1. Run $\mathsf{Share}(n, t, b)$: Apply the (t, n)- secret sharing scheme to output n shares $\{s_i\}_{i=1}^n$.
2. Run $\mathsf{TREnc.Enc}$:
   - If aux is specified, set $\mathsf{lk}_{ki} = \mathsf{aux}_{ki}$ for $k = \{1, 2\}, i \in [n]$. Otherwise, $\mathsf{lk}_{ki} = \mathsf{TREnc.LGen}(\mathsf{PK}_k)$.
   - Compute $\mathcal{C}_i = \mathsf{TREnc.LEnc}(\mathsf{PK}_1, \mathsf{lk}_{1i}, G^{s_i})$, $\hat{\mathcal{C}}_i = \mathsf{TREnc.LEnc}$ $(\mathsf{PK}_2, \mathsf{lk}_{2i}, \hat{G}^{s_i})$, where the stripped CPA parts respectively are given by $\boldsymbol{c}_i = \mathsf{TREnc.Strip}(\mathsf{PK}_1, \mathcal{C}_i)$, $\hat{\boldsymbol{c}}_i = \mathsf{TREnc.Strip}(\mathsf{PK}_2, \hat{\mathcal{C}}_i)$ such that
   $$\boldsymbol{c}_i = (c_{0i}, c_{1i}, c_{2i}) = (G^{s_i} f^{\alpha_i}, g^{\alpha_i}, h^{\alpha_i})$$
   $$\hat{\boldsymbol{c}}_i = (\hat{c}_{0i}, \hat{c}_{1i}, \hat{c}_{2i}) = (\hat{G}^{s_i} \hat{f}^{\beta_i}, \hat{g}^{\beta_i}, \hat{h}^{\beta_i})$$
   with $(f, g, h) = \mathsf{TREnc.Strip}(\mathsf{PK}_1)$, $(\hat{f}, \hat{g}, \hat{h}) = \mathsf{TREnc.Strip}(\mathsf{PK}_2)$, and $\alpha_i, \beta_i \leftarrow\$ \mathbb{Z}_p$.

SetupElection (λ)

$(\mathsf{SK}_1, \mathsf{PK}_1) \leftarrow \mathsf{TREnc.Gen}(1^\lambda)$
$(\mathsf{SK}_2, \mathsf{PK}_2) \leftarrow \mathsf{TREnc.Gen}(1^\lambda)$
$\sigma \leftarrow \mathsf{PrfSetup}(1^\lambda); \mathsf{crs} \leftarrow \mathsf{GSSetup}(1^\lambda)$
$\mathsf{SK} = (\mathsf{SK}_1, \mathsf{SK}_2)$
return $\mathsf{PK} = (\mathsf{PK}_1, \mathsf{PK}_2, \sigma, \mathsf{crs})$

Vote(id, v, l, [, aux])

$\mathsf{v} = \Sigma_j b_j 2^j$ for $b_j \in \{0,1\}, j \in [l]$
for $j = 1$ to l do
  $\{b_{ji}\}_{i=1}^n \leftarrow \mathsf{Share}(\mathsf{PK}, t, b_j)$
if aux is empty **for** $j = 1$ to l do
  for $i = 1$ to n do
    $\mathsf{lk}_{ji} \leftarrow\$ \mathsf{TREnc.LGen}(\mathsf{PK})$
    $\hat{\mathsf{lk}}_{ji} \leftarrow\$ \mathsf{TREnc.LGen}(\mathsf{PK})$
else $\{\mathsf{lk}_{ji}, \hat{\mathsf{lk}}_{ji}\}_{j,i} \leftarrow \mathsf{aux}$
for $j = 1$ to l do
  for $i = 1$ to n do
    $\mathcal{C}_{ji} \leftarrow \mathsf{TREnc.LEnc}(\mathsf{PK}_1, \mathsf{lk}_{ji}, b_{ji})$
    $\hat{\mathcal{C}}_{ji} \leftarrow \mathsf{TREnc.LEnc}(\mathsf{PK}_2, \hat{\mathsf{lk}}_{ji}, b_{ji})$
    $\boldsymbol{\theta}_{ji} \leftarrow \mathsf{GSProve}(\mathsf{PK}, \mathcal{C}_{ji}, \hat{\mathcal{C}}_{ji})$
    $\mathsf{b}_{ji} = (\mathcal{C}_{ji}, \hat{\mathcal{C}}_{ji}, \boldsymbol{\theta}_{ji})$
return $\mathsf{b} = \{\mathsf{b}_{ji}\}_{j,i}^{l,n}$

ProcessBallot(b_i)

for $j = 1$ to l do
  $\mathcal{C}'_{ji} \leftarrow \mathsf{TREnc.Rand}(\mathsf{PK}_1, \mathcal{C}_{ji})$
  $\hat{\mathcal{C}}'_{ji} \leftarrow \mathsf{TREnc.Rand}(\mathsf{PK}_2, \hat{\mathcal{C}}_{ji})$
  $\boldsymbol{\theta}'_{ji} \leftarrow \mathsf{GSRand}(\mathsf{PK}, \hat{\mathcal{C}}_{ji}, \mathcal{C}_{ji}, \boldsymbol{\theta}_{ji})$
  $\mathsf{b}_{ji} = (\mathcal{C}'_{ji}, \hat{\mathcal{C}}'_{ji}, \boldsymbol{\theta}'_{ji})$
return $\mathsf{b}'_i = \{\mathsf{b}_{ji}\}_{j \in [l]}$

Valid(BB, b)

if $\exists \mathsf{b}' \in \mathsf{BB} \wedge \exists \tau'_i \subset \mathsf{TraceBallot}(\mathsf{b}')$:
$\tau'_i \subset \mathsf{TraceBallot}(\mathsf{b})$ **then return** $\perp$
for $j = 1$ to l do
  $k = 0$
  for $i = 1$ up to n do
    if $\mathsf{GSVerify}(\mathsf{PK}, \boldsymbol{\theta}_{ji}) = 1$
      $\wedge\ \mathsf{TREnc.Ver}(\mathsf{PK}_1, \mathcal{C}_{ji}) = 1$
      $\wedge\ \mathsf{TREnc.Ver}(\mathsf{PK}_2, \hat{\mathcal{C}}_{ji}) = 1$
     then $k \leftarrow k + 1$
  if $k < t$ **then return** 0
return 1

TraceBallot(b)

$\tau_{ji} \leftarrow (\mathsf{TREnc.Trace}(\mathcal{C}_{ji}), \mathsf{TREnc.Trace}(\hat{\mathcal{C}}_{ji}))$
return $\tau = \{\tau_{ji}\}_{j,i}^{l,n}$

PreTally (BB, SK, b)

if $\mathsf{Valid}(\mathsf{BB}, \mathsf{b}) = 0$ **then return** 0
for $j = 1$ to l do
  for $i = 1$ up to n do
    $\boldsymbol{c}_{ji} \leftarrow \mathsf{TREnc.Strip}(\mathsf{PK}_1, \mathcal{C}_{ji})$
    $\hat{\boldsymbol{c}}_{ji} \leftarrow \mathsf{TREnc.Strip}(\mathsf{PK}_2, \hat{\mathcal{C}}_{ji})$
  $\boldsymbol{c}_j \leftarrow \mathsf{Combine}(n, t, \{\boldsymbol{c}_{ji}\}_{i=1}^{\leq n})$
  $\hat{\boldsymbol{c}}_j \leftarrow \mathsf{Combine}(n, t, \{\hat{\boldsymbol{c}}_{ji}\}_{i=1}^{\leq n})$
$\boldsymbol{C} = \{(\boldsymbol{c}_j, \hat{\boldsymbol{c}}_j)\}_{j=1}^l$
return MPC-$\mathsf{HomoRand}(\mathsf{PK}, \mathsf{SK}, \boldsymbol{C})$

VerifyVote(PBB, τ)

if $\exists \mathsf{b} \in \mathsf{PBB} : \mathsf{Valid}(\mathsf{b}) \wedge \tau == \mathsf{TraceBallot}(\mathsf{b})$
then return 1 **else return** 0

Fig. 3: HomoRand instantiation of our voting scheme.

3. Ensure encrypted messages $G^{s_i}, \hat{G}^{s_i}$ share the same exponent by computing the following for each $i = 1, \ldots, n$:

   – Commit to the scalars s_i, α_i in $\mathcal{C}_i$ by $\hat{\boldsymbol{C}}_{2,i} = \boldsymbol{\varphi}^{s_i}\boldsymbol{\psi}^{\rho_{2i}}$, $\hat{\boldsymbol{C}}_{\alpha,i} = \boldsymbol{\varphi}^{\alpha_i}\boldsymbol{\psi}^{\rho_{\alpha i}}$ with $\rho_{2i}, \rho_{\alpha i} \leftarrow\$ \mathbb{Z}_p$.
   – Commit to the scalars s_i, β_i in $\hat{\mathcal{C}}_i$ by $\boldsymbol{C}_{1,i} = \boldsymbol{u}_1^{s_i}\boldsymbol{u}_2^{\rho_{1i}}$, $\boldsymbol{C}_{\beta,i} = \boldsymbol{u}_1^{\beta_i}\boldsymbol{u}_2^{\rho_{\beta i}}$, with $\rho_{1i}, \rho_{\beta i} \leftarrow\$ \mathbb{Z}_p$.

– Pick $t \leftarrow\!\$\, \mathbb{Z}_p$ and compute the GS proofs $\boldsymbol{\theta}_i = \{\theta_{ji}\}_{j=1}^{8} = (G^{\rho_{2i}} f^{\rho_{\alpha i}}, g^{\rho_{\alpha i}}, h^{\rho_{\alpha i}}, \hat{G}^{\rho_{1i}} \hat{f}^{\rho_{\beta i}}, \hat{g}^{\rho_{\beta i}}, \hat{h}^{\rho_{\beta i}}, \boldsymbol{\varphi}^{\rho_{1i}} \boldsymbol{\psi}^{t}, \boldsymbol{u}_1^{\rho_{2i}} \boldsymbol{u}_2^{t})$:

$$E_2(c_{0i}, \boldsymbol{\varphi}) E_2(\theta_{1i}, \boldsymbol{\psi}) = E_2(G, \hat{\boldsymbol{C}}_{2,i}) E_2(f, \hat{\boldsymbol{C}}_{\alpha,i}) \tag{1}$$
$$E_2(c_{1i}, \boldsymbol{\varphi}) E_2(\theta_{2i}, \boldsymbol{\psi}) = E_2(g, \hat{\boldsymbol{C}}_{\alpha,i}) \tag{2}$$
$$E_2(c_{2i}, \boldsymbol{\varphi}) E_2(\theta_{3i}, \boldsymbol{\psi}) = E_2(h, \hat{\boldsymbol{C}}_{\alpha,i}) \tag{3}$$
$$E_1(\boldsymbol{u}_1, \hat{c}_{0i}) E_1(\boldsymbol{u}_2, \theta_{4i}) = E_1(\boldsymbol{C}_{1,i}, \hat{G}) E_1(\boldsymbol{C}_{\beta,i}, \hat{f}) \tag{4}$$
$$E_1(\boldsymbol{u}_1, \hat{c}_{1i}) E_1(\boldsymbol{u}_2, \theta_{5i}) = E_1(\boldsymbol{C}_{\beta,i}, \hat{g}) \tag{5}$$
$$E_1(\boldsymbol{u}_1, \hat{c}_{2i}) E_1(\boldsymbol{u}_2, \theta_{6i}) = E_1(\boldsymbol{C}_{\beta,i}, \hat{h}) \tag{6}$$
$$E(\boldsymbol{u}_1, \hat{\boldsymbol{C}}_{2,i}) E(\boldsymbol{u}_2, \boldsymbol{\theta}_{7i}) = E(\boldsymbol{C}_{1,i}, \boldsymbol{\varphi}) E(\boldsymbol{\theta}_{8i}, \boldsymbol{\psi}) \tag{7}$$

4. Set $(\mathcal{C}_i, \hat{\mathcal{C}}_i, \hat{\boldsymbol{C}}_{2,i}, \hat{\boldsymbol{C}}_{\alpha,i}, \boldsymbol{C}_{1,i}, \boldsymbol{C}_{\beta,i}, \boldsymbol{\theta}_i)$ as output of $\mathsf{Enc}(\mathsf{PK}, b, \mathsf{aux})$.

Denote $\mathsf{CT}_j = \mathsf{Enc}(\mathsf{PK}, b_j, \mathsf{aux}_j)$ for $\mathsf{aux}_j \in \mathsf{aux}$, the voter sends the ciphertext $\mathsf{CT} = \{\mathsf{CT}_j\}_{j=1}^{l}$ to corresponding randomizers, where $\mathsf{CT}_j = \{\mathsf{CT}_{ji}\}_{i=1}^{n}$ and $\mathsf{CT}_{ji} = (\mathcal{C}_{ji}, \hat{\mathcal{C}}_{ji}, \hat{\boldsymbol{C}}_{2,ji}, \hat{\boldsymbol{C}}_{\alpha,ji}, \boldsymbol{C}_{1,ji}, \boldsymbol{C}_{\beta,ji}, \boldsymbol{\theta}_{ji})$ as previously described.

$\underline{\mathsf{ProcessBallot}(\mathsf{PK}, \mathsf{CT}_i)}$: If PK or $\mathsf{CT}_i = \{\mathsf{CT}_{ji}\}_{j\in[l]}$ do not parse properly, abort. Otherwise, a randomizer conducts the following steps for any $j \in [l]$:

– Compute $\mathcal{C}'_{ji} = \mathsf{TREnc.Rand}(\mathsf{PK}_1, \mathcal{C}_{ji})$ with $\boldsymbol{c}'_{ji} = \mathsf{Strip}(\mathsf{PK}_1, \mathcal{C}'_{ji})$ and $\hat{\mathcal{C}}'_{ji} = \mathsf{TREnc.Rand}(\mathsf{PK}_2, \hat{\mathcal{C}}_{ji})$, where $\hat{\boldsymbol{c}}'_{ji} = \mathsf{Strip}(\mathsf{PK}_2, \hat{\mathcal{C}}'_{ji})$ such that

$$\boldsymbol{c}'_{ji} = (c'_{0ji}, c'_{1ji}, c'_{2ji}) = (c_{0ji} \cdot f^{\alpha'_{ji}}, c_{1ji} \cdot g^{\alpha'_{ji}}, c_{2ji} \cdot h^{\alpha'_{ji}})$$
$$\hat{\boldsymbol{c}}'_{ji} = (\hat{c}'_{0ji}, \hat{c}'_{1ji}, \hat{c}'_{2ji}) = (\hat{c}_{0ji} \cdot \hat{f}^{\beta'_{ji}}, \hat{c}_{1ji} \cdot \hat{g}^{\beta'_{ji}}, \hat{c}_{2ji} \cdot \hat{h}^{\beta'_{ji}}),$$

with $\alpha'_{ji}, \beta'_{ji} \leftarrow\!\$\, \mathbb{Z}_p$.

– Pick $\rho'_{1ji}, \rho'_{\beta ji}, \rho'_{2ji}, \rho'_{\alpha ji}, t'_{ji} \leftarrow\!\$\, \mathbb{Z}_p$ and adapt the GS commitments and proofs:
  - Compute $\hat{\boldsymbol{C}}'_{2,ji} = \hat{\boldsymbol{C}}_{2,ji} \cdot \boldsymbol{\psi}^{\rho'_{2ji}}$, $\hat{\boldsymbol{C}}'_{\alpha,ji} = \hat{\boldsymbol{C}}_{\alpha,ji} \cdot \boldsymbol{\varphi}^{\alpha'_{ji}} \boldsymbol{\psi}^{\rho_{\alpha ji'}}$, $\boldsymbol{C}'_{1,ji} = \boldsymbol{C}_{1,ji} \cdot \boldsymbol{u}_2^{\rho'_{1ji}}$, and $\boldsymbol{C}'_{\beta,ji} = \boldsymbol{C}_{\beta,ji} \cdot \boldsymbol{u}_1^{\beta'_{ji}} \boldsymbol{u}_2^{\rho'_{\beta ji}}$.
  - Update the proofs $\boldsymbol{\theta}'_{ji} = (\theta_{1ji} \cdot G^{\rho'_{2ji}} f^{\rho'_{\alpha ji}}, \theta_{2ji} \cdot g^{\rho'_{\alpha ji}}, \theta_{3ji} \cdot h^{\rho'_{\alpha ji}}, \theta_{4ji} \cdot \hat{G}^{\rho'_{1ji}} \hat{f}^{\rho'_{\beta ji}}, \theta_{6ji} \cdot \hat{g}^{\rho'_{\beta ji}}, \theta_{7ji} \cdot \hat{h}^{\rho'_{\beta ji}}, \boldsymbol{\theta}_{8ji} \cdot \boldsymbol{\varphi}^{\rho'_{1ji}} \boldsymbol{\psi}^{t'_{ji}}, \boldsymbol{\theta}_{9ji} \cdot \boldsymbol{u}_1^{\rho'_{2ji}} \boldsymbol{u}_2^{t'_{ji}})$.

– Publish $\mathsf{CT}'_{ji} = (\mathcal{C}'_{ji}, \hat{\mathcal{C}}'_{ji}, \hat{\boldsymbol{C}}'_{2,ji}, \hat{\boldsymbol{C}}'_{\alpha,ji}, \boldsymbol{C}'_{1,ji}, \boldsymbol{C}'_{\beta,ji}, \boldsymbol{\theta}'_{ji})$.

$\underline{\mathsf{Valid}(\mathsf{PK}, \mathsf{CT}')}$: Abort and return 0 if PK or CT' is not parsed properly. Return 1 if there exists a subset $I \subset [n]$ such that $|I| \geq t$ and for all $i \in I$:

– $\mathsf{TREnc.Ver}(\mathsf{PK}_1, \mathcal{C}'_{ji}) = 1$ and $\mathsf{TREnc.Ver}(\mathsf{PK}_2, \hat{\mathcal{C}}'_{ji}) = 1$, and
– The equations 1- 7 hold.

Otherwise, return 0. If $\mathsf{Valid}(\mathsf{PK}, \mathsf{CT}') = 1$, update $\mathsf{CT}' \leftarrow \{\mathsf{CT}'_{ji}\}_{j,i}$ for $i \in I$ and all $j \in [l]$.

$\underline{\mathsf{PreTally}(\mathsf{SK}, \mathsf{CT}')}$: Abort and output 0 if PK or CT' does not parse properly or $\mathsf{Valid}(\mathsf{PK}, \mathsf{CT}') = 0$. Otherwise, conduct the following steps:

1. For each $\mathsf{CT}'_{ji} \in \mathsf{CT}'_j$, run $\mathsf{Strip}(\mathsf{PK}, (\mathcal{C}'_{ji}, \hat{\mathcal{C}}'_{ji}))$ to only extract the CPA components, denoted as $\mathsf{CPA}(\mathsf{CT}'_{ji})$.

2. Run $\mathsf{Combine}(n, t, \{\mathsf{CPA}(\mathsf{CT}'_{ji})\}_{i=1}^{|I|})$ using Lagrange interpolation for $j \in [l]$. Since the CPA parts in TREnc is homomorphic, this results in $(\boldsymbol{c}'_j, \hat{\boldsymbol{c}}_j{}')$, where

$$\boldsymbol{c}'_j = (c'_{0j}, c'_{1j}, c'_{2j}) = (G^{b_j} f^{\alpha_j + \alpha'_j}, g^{\alpha_j + \alpha'_j}, h^{\alpha_j + \alpha'_j})$$
$$\hat{\boldsymbol{c}}'_j = (\hat{c}'_{0j}, \hat{c}'_{1j}, \hat{c}'_{2j}) = (\hat{G}^{bj} \hat{f}^{\beta_j + \beta'_j}, \hat{g}^{\beta_j + \beta'_j}, \hat{h}^{\beta_j + \beta'_j}),$$

where $\alpha_j + \alpha'_j = \sum_{i=1}^{|I|} \Lambda_{ji}(\alpha_{ji} + \alpha'_{ji})$ and $\beta_j + \beta'_j = \sum_{i=1}^{|I|} \Lambda_{ji}(\beta_{ji} + \beta'_{ji})$. Set $\boldsymbol{C}' = \{\boldsymbol{C}'_j\}_{j=1}^{l}$ where $\boldsymbol{C}'_j = (\boldsymbol{c}'_j, \hat{\boldsymbol{c}}_j{}')$.
3. Run MPC-$\mathsf{HomoRand}(\mathsf{PK}, \boldsymbol{C}')$ (Algorithms 2 and 3) as described in Section A.2.

$\mathsf{PreTally}$ returns 1 if and only if v is valid. Finally, the talliers (or anyone) compute the ciphertext $\mathsf{ct} = \prod_{j=1}^{l} (\boldsymbol{c}'_j)^{2^j}$, and forwarded for tallying according to the standard procedure.