OMIX: Offline Mixing for Self-Tallying Elections

Sébastien Canard[1], Liam Medley[2], Duy Nguyen[1], and Duong Hieu Phan[1]

[1] Telecom Paris, Institut Polytechnique de Paris, France
{sebastien.canard,dinh.nguyen,hieu.phan}@telecom-paris.fr
[2] lmedley@sky.com

Abstract. In e-voting, a mix-net is a widely used system for shuffling votes in order to guarantee voter privacy. The core principle is as follows: (1) N voters cast their votes; (2) a sequence of mix-servers shuffles the N encrypted ballots, thereby randomly permuting the underlying votes; and (3) trustees jointly decrypt the shuffled ballots to compute the tally.
This approach, however, relies heavily on trusted parties *after* the voting phase and raises serious robustness issues. For example, what happens if the mix-servers disrupt the protocol, or if trustees lose the decryption key during the tallying phase? A recent incident in the IACR board election illustrates this problem: one of the three trustees involuntarily lost their key, requiring the entire community to restart the vote.
We introduce OMIX, an offline-mixing architecture for self-tallying elections: trustees prepare public functional decryption keys embedding a hidden random permutation before voting, voters then cast encrypted ballots, and anyone can compute the shuffled tally without further trustee participation. Function-hiding inner-product functional encryption supplies the key mechanism, and a voter-assisted correction phase accommodates dropouts.
The main technical component is a verifiable offline shuffle of a matrix of encrypted functional keys. We extend linearly homomorphic signatures with tag-binding and two-dimensional tags: row tags support hidden row permutations, while column tags bind entries to their respective voters. In the certified-input model, these properties allow trustees to certify matrix transformations, with final proof size and verification cost independent of the number of mix-servers.
Together, these components enable distributed preparation and public certification of permutation keys within a client-lightweight voting architecture. Each voter registers and casts a ballot with computation and communication independent of N, while preprocessing per trustee, public helper-key size, and tallying work are quadratic. This establishes a foundation for self-tallying elections with offline mixing and opens the way to further improvements in system-wide efficiency.

Keywords: voting · mix-net · self-tallying · functional encryption · linearly homomorphic signature · distributed joint computation

Table of Contents

1 Introduction . . . 3
1.1 Our Results . . . 4
1.2 Technical Overview . . . 7
1.3 Related Work . . . 11
1.4 Organisation of the Paper . . . 12
2 Notation and Standard Primitives . . . 13
2.1 Notation . . . 13
2.2 Groups . . . 13
2.3 Asymmetric Pairing Groups . . . 14
2.4 Public Key Encryption . . . 14
2.5 Non-Interactive Zero-Knowledge Proofs . . . 15
3 Voting Definitions . . . 15
3.1 Voting Syntax . . . 15
3.2 Corrective Fault Tolerance . . . 17
3.3 Client-Side Lightweightness . . . 17
3.4 Ballot Privacy . . . 18
3.5 Verifiability . . . 19
3.6 Other Notions . . . 20
4 Voting with Functional Encryption . . . 20
4.1 Preliminaries on Functional Encryption . . . 20
4.2 A Warm-up Protocol . . . 21
4.3 Ballot Privacy . . . 23
4.4 Tallying Efficiency . . . 25
5 Voting with an Offline Mix-Net . . . 25
5.1 OMIX-based Voting Framework . . . 25
5.2 Correctness, Corrective Fault Tolerance, and Client-Side Lightweightness . . . 26
5.3 Universal Verifiability . . . 29
5.4 Ballot Privacy . . . 30
6 OMIX from Linearly Homomorphic Signatures . . . 33
6.1 Preliminaries on Linearly Homomorphic Signatures . . . 33
6.2 Enhancing Linearly Homomorphic Signatures . . . 34
6.3 Construction of 2-Tag-LHS . . . 36
6.4 OMIX Construction . . . 38
7 Security of Offline Protocols and Their Integration . . . 39
7.1 Auxiliary Security Notions for Offline Protocols . . . 39
7.2 Deferred Proofs for OMIX . . . 43
7.3 OPAD for Randomness Generation . . . 46
7.4 Integrating Offline Protocols in Voting . . . 50

1 Introduction

Privacy is a fundamental goal of cryptography and a critical requirement for many applications, particularly in secure electronic voting (e-voting). One of the key mechanisms to ensure voter privacy is a valid *shuffle* protocol, which ensures that ciphertexts (encrypted votes) are shuffled before they are decrypted. In such a protocol, the input is a set of encrypted votes, and the output is the corresponding plaintexts in a permuted order, such that it is impossible to associate any input ciphertext with its corresponding plaintext.

A widely used approach to achieve this shuffle is the *mix-net*. In a mix-net, a group of mix servers sequentially shuffle and re-encrypt (or decrypt) the ciphertexts, ensuring that as long as one server is honest, the initial order of the plaintexts remains private. Since its introduction by Chaum in 1981 [Cha81], the mix-net has proven to be a valuable cryptographic primitive, particularly because it supports elections with arbitrary voting formats.

Consider the standard approach to mix-net-based voting: after ballots are cast, a group of trusted authorities are responsible for mixing and decrypting the ciphertexts in order to compute the final result. Many existing works explore how to verify that these authorities have fulfilled their duties correctly, typically through zero-knowledge proofs of correct shuffling and decryption.

However, how to prevent these authorities from disrupting the election remains a fundamental challenge. For instance, an authority may refuse to participate, abort midway, or submit an invalid proof. Any of these actions could obstruct the tallying process and ultimately spoil the election result. This risk is particularly relevant in practice, as modern elections often release exit polls before official results, potentially motivating a politically influenced authority to disrupt the protocol upon anticipating an unfavourable outcome.

More generally, even non-malicious failures can have severe consequences. A recent incident in the IACR board election illustrates this issue concretely: one of the election trustees involuntarily lost their decryption key share, making it impossible to complete the tallying process and forcing the election to restart from scratch [Int25].

To address this, we introduce a new voting framework based on an *offline mix-net*, in which any party can verify that the tasks assigned to mix-servers and decryption trustees are correctly completed during an offline phase, prior to voting. More importantly, the online tallying process can then be carried out entirely in public by any observer, without further trustee participation. In other words, our framework eliminates dependence on trusted authorities holding secret keys for computing the result and shifts this capability to the public.

The notion that an election result can be computed publicly from the voters' ballots alone was first introduced under the term *self-tallying* by Kiayias and Yung in [KY02]. Whilst self-tallying has been shown to be compatible with both homomorphic aggregation- and mix-net-based systems [KY02, Gro04a], extending it to mix-net-based elections with *client-side lightweightness*, meaning that each voter's computational cost is independent of the total number of voters, remains an open problem. In other words, our work addresses the following question:

> *Is it possible to construct a voting system that is simultaneously client-side lightweight, mix-net-based, and self-tallying?*

It is worth noting that prior works on self-tallying elections usually aim for *maximal ballot privacy*[3], where no trusted authority is involved and each voter fully controls the privacy of their ballot. Although this provides stronger privacy guarantees, it often requires voters themselves to participate in the mixing process; for example, voters may jointly shuffle ballots as part of the voting phase [Gro04a]. As a result, each voter's computation and communication costs grow with the number of voters, making such approaches difficult to scale. On the other hand, relying on anonymous broadcast channels, as in [AAG+25], may in practice require anonymizing entities like mix-servers during ballot transmission to the bulletin board, i.e., after voters decide on and submit their votes.

We observe that the main obstacle to achieving client-side lightweightness is that existing approaches do not explicitly separate the two roles of mix-servers: contributing random permutations and applying these permutations to voters' ballots. To address this, we introduce a new self-tallying architecture in which the contribution of random permutations by servers is performed entirely in an offline phase before voting. The jointly generated permutation is then embedded privately into a public decryption key that reveals only the shuffled votes when applied to the voters' ballots. This separation ultimately enables client-side lightweightness.

[3] The same notion was introduced under the name *perfect ballot secrecy* in [KY02].

Another appealing aspect of our construction, particularly for e-voting settings, is that it relies largely on bilinear-group operations, meaning that the construction does not exploit the representing strings of the underlying cryptographic primitives. This suggests a plausibly practical approach.

Achieving this, however, requires addressing a novel mixing structure induced by offline mix-nets for verifiability. In particular, we design a verification mechanism for this structure without relying on general-purpose zero-knowledge proofs, for which no dedicated shuffle verification technique was previously known. Concretely, we achieve an offline-verifiable mix-net in which *the proof size and the online verification cost*[4] *are independent of the number of mix-servers.*

1.1 Our Results

Our contributions have three complementary aspects:

- **Voting architecture.** We prepare the hidden permutation and its public functional decryption keys before voting, so that, after successful preprocessing, tallying no longer requires trustee participation. A voter-assisted correction phase accommodates dropouts.
- **Verifiable offline matrix shuffle.** We extend randomisable-tag linearly homomorphic signatures with tag-binding and two-dimensional tags to certify row shuffles of encrypted functional-key matrices while preserving column bindings. This provides the verification mechanism for distributed permutation preparation in the certified-input model.
- **Client-side lightweightness.** Registration and ballot casting have computation and communication independent of the total number of voters. This offloads work from voters to quadratic preprocessing per server, public helper-key storage, and tallying.

The warm-up protocol explains how functional decryption reveals a shuffled vote vector; the full construction combines this mechanism with the voting architecture and certified distributed preprocessing.

Mix-net-based self-tallying with client-lightweightness We generate a public decryption key during the *offline phase* so that votes are revealed only in a randomly permuted order. Assuming honest-but-curious voters, the resulting self-tallying election is illustrated in Figure 1. This is in contrast to the standard approach shown in Figure 2, where the mix-servers and decryption trustees must handle ballots directly, and no self-tallying property is achieved.

Our constructions show that the key generation for self-tallying elections as in Figure 1 can be achieved through an abstract application of *functional encryption* for function-hiding inner products, a primitive that allows only inner-product evaluations of encrypted data to be revealed via functional decryption keys. Furthermore, we obtain a voting protocol proceeding as follows:

- **Registration phase:** Each voter locally runs a key-generation procedure and non-interactively registers its own functional decryption keys. Assuming n voters and m authorities, the computational and communication costs incurred by each voter are independent of n, i.e., $O(1)$.
- **Preprocessing phase:** Authorities (including mix-servers) preprocess the voters' functional decryption keys into a public decryption key for a hidden permutation, whose size is $O(n^2)$. Through dedicated protocols, each server distributively contributes its own permutation to the registered keys. This phase incurs $O(n^2)$ computation and communication costs per server.
- **Voting phase:** Each voter simply encrypts and casts its vote with $O(1)$ computational and communication costs.
- **Tallying phase:** Tallying reduces to a public decryption procedure in which the public decryption key is applied to the ballots to reveal the shuffled result. This procedure incurs $O(n^2)$ computational cost.
- **Fault correction:** If registered voters do not submit valid ballots, every voter whose ballot is included in the tally locally generates and publishes a correction key. Recovery requires all these voters to complete the correction phase honestly. The cost incurred by each voter remains $O(1)$.

It is important to note that a self-tallying scheme cannot simultaneously guarantee both robustness and privacy (see Proposition 1 in [KY02]), because if any casual party can tally an arbitrary subset of encrypted votes and obtain the correct result, individual votes will be revealed. Consequently, the tally expects all ballots from registered voters. To manage dropout voters who fail to cast their ballots, the tally might require actions from the remaining honest voters, a property known as *corrective fault*

[4] If the set of mix-servers is fixed in advance, a verification key can be pre-processed.

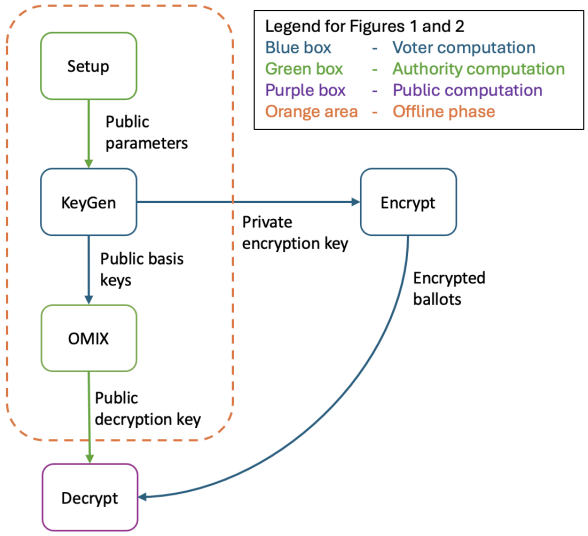


Fig. 1: Our new approach to voting with mix-nets. We separate the permutation generation and shuffling steps, which are executed simultaneously during standard mixing. In an offline OMIX, mix-servers embed a hidden permutation into a public decryption key. Using this key, any party can perform an oblivious shuffle of votes via decryption, meaning the shuffle is carried out without knowledge of the permutation.

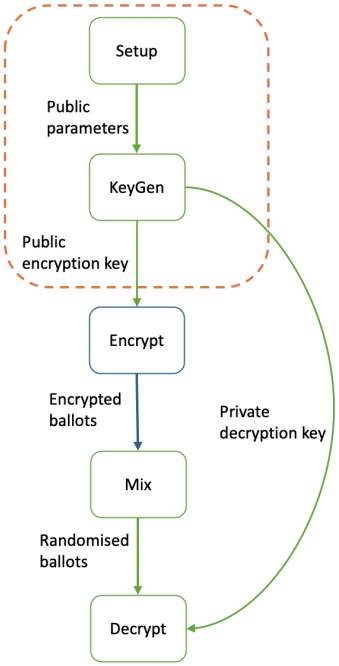


Fig. 2: The standard approach to voting using a mix-net. The mixing and decrypting steps require the respective authorities to be online during the tallying phase, where encrypted votes are mixed and decrypted.

tolerance [KY02]. Therefore, regarding voter round complexity, at most three casts from voters are required in our protocol: one for registering, one for voting, and one for handling dropout voters.

Once preprocessing succeeds, trustees need not remain available for tallying. Without dropouts, no further voter action is needed after casting; with dropouts, all participating voters supply correcting keys. This separates trustee availability from the voter-assisted recovery inherent in our corrective-fault-tolerance guarantee.

A comparison with existing self-tallying systems is presented in Figures 3 and 4.

Voting System	Result Function on Votes	Maximal Ballot Privacy	†Client-Side Lightweight-ness	Corrective Fault Tolerance	No Mix-servers after Ballot Creation	No Decrypting Trustees after Ballot Creation
KY1	Aggregate	●	○	●	N/A	●
KY2	Aggregate	○	◐	●	N/A	◑
Groth1	Aggregate	●	○	●	N/A	●
Groth2	Shuffle	●	○	○	●	●
E-cclesia [AAG+25]	Shuffle	◐	◑	N/A	○	●
QV-net [ZZH+25]	Aggregate	●	○	N/A	N/A	●
OMIX	Shuffle	◐	◑	●	●	●

Fig. 3: Comparison with relevant self-tallying voting systems. We highlight [ZZH+25] and [AAG+25] as the current state-of-the-art systems for homomorphic aggregation and mix-nets, respectively. The schemes in the table are labeled as KY1 and KY2 for the two schemes from [KY02] (see Sections 3 and 4.3, respectively), and Groth1 and Groth2 for those from [Gro04a] (see Sections 2 and 3, respectively). The symbols ● and ○ indicate that the corresponding property is achieved and not achieved, respectively. The symbols ◐ and ◑ denote that the two properties cannot be achieved simultaneously. "N/A" means the property is not applicable. †Client-side lightweightness is defined in terms of each voter's efficiency up until the end of the voting phase, while the tallying can be delegated to computationally powerful, untrusted servers.

As this is primarily a conceptual work, we leave the development of more efficient tallying mechanisms, stronger security guarantees, and practical implementations to future work.

Verifiable offline mix-nets The offline preprocessing phase introduces a novel mixing structure for mix-servers, namely a matrix of encrypted functional keys associated with voters, where each mix-server sequentially re-randomises the encryption layer and permutes the rows to embed its own permutation. We aim to achieve a mix-net proof whose size and verification cost are independent of the number of mix-servers, while reducing verification to simple per-ciphertext checks. To this end, and as a contribution of independent interest, we extend *linearly homomorphic signatures* (LHS) with rerandomisable tags to support matrix shuffles, and apply this construction to obtain a verifiable mix-net in the spirit of [HPP20].

At a high level, LHS is structure-preserving and can serve as a malleable proof system for the language of ciphertext re-randomisation. Each tag is associated with a specific ciphertext, thereby defining a distinct statement. The re-randomisability of the tags ensures unlinkability between the original and shuffled ciphertexts. Moreover, due to the malleability property, each mix-server can derive its proof from the proof generated by the previous server. Consequently, it suffices to verify only the re-randomised and shuffled ciphertexts output by the final mix-server.

Several security challenges arise when adapting LHS to our setting, since mix-servers in our framework shuffle rows of a ciphertext matrix rather than entries of a ciphertext vector. In particular, there are two issues: (1) existing LHS schemes do not guarantee that ciphertexts validated under the same re-randomised tag originate from a common source tag, and (2) existing tagging structures do not naturally support two-dimensional tags for matrix settings, where row and column positions must both be bound while still allowing rerandomization of the row component.

To address these issues, we construct an LHS scheme with two simultaneous enhancements: *a stronger tag-binding unforgeability notion* and *a new two-dimensional tagging structure*. Together, these yield a

Voting System	No Post-Voting Authority for Fairness	††Fully Synchronous Voting	Tallying Efficiency for n Voters	†††Computational/ Primitive Assumption
KY1	○	●	$O(n)$	DLOG
KY2	○	●	$O(n)$	DLOG
Groth1	○	○	$O(n)$	DDH
Groth2	○	○	$O(n)$	DDH
E-cclesia [AAG+25]	●	●	O(n)	Time-lock encryption, anonymous broadcast channel
QV-net [ZZH+25]	○	●	$O(n)$	DDH
OMIX	○	●	$O(n^2)$	Function-hiding IPFE, SXDH

Fig. 4: Comparison with relevant self-tallying voting systems (continued). ††Fully synchronous voting means that voters can independently cast their ballots at the same time without causing errors in the protocol. †††Since the assumptions underlying verification mechanisms may vary depending on the instantiation of, e.g., zero-knowledge proof systems, we report only the assumptions required for semi-honest security, where all parties are assumed to be honest-but-curious.

novel LHS construction in the bilinear generic group model that satisfies the construction requirements of our offline mix-net.

1.2 Technical Overview

We first present a self-tallying protocol with only a single trustee to illustrate our core components. Then, we present an offline mix-net for the multi-server setting (OMIX) to distribute trust away from a single trustee.

Let $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ be an asymmetric bilinear group of prime order p such that for any $a \in \mathbb{Z}_p$ and $s \in \{1, 2\}$, $[\![a]\!]_s = a \cdot P_s$. Furthermore, $[\![a]\!]_T = a \cdot e(P_1, P_2)$. Here, $\cdot$ denotes scalar multiplication, whereas e denotes the pairing; products of scalars are written by juxtaposition. Our construction relies on an abstract use of functional encryption for function-hiding inner products (FH-IPFE) over bilinear groups, together with its homomorphic functional keys.

Following [BJK15], an FH-IPFE scheme over prime-order bilinear groups consists of four algorithms:

$\mathsf{SetUp}(1^\lambda, 1^\rho) \to (\mathsf{pp}, \mathsf{sk})$: On input a security parameter λ and a length ρ, the setup algorithm generates public parameters pp and a secret key sk.

$\mathsf{DKGen}(\mathsf{sk}, [\![\boldsymbol{y}]\!]_1) \to \mathsf{dk}_{\boldsymbol{y}}$: On input a secret key sk and a function that is represented by $[\![\boldsymbol{y}]\!]_1 \in \mathbb{G}_1^\rho$, it outputs a functional (decryption) key $\mathsf{dk}_{\boldsymbol{y}}$.

$\mathsf{Enc}(\mathsf{sk}, [\![\boldsymbol{x}]\!]_2) \to \mathsf{ct}_{\boldsymbol{x}}$: On input a secret key sk and a message $[\![\boldsymbol{x}]\!]_2 \in \mathbb{G}_2^\rho$, it outputs a ciphertext $\mathsf{ct}_{\boldsymbol{x}}$.

$\mathsf{Dec}(\mathsf{dk}, \mathsf{ct}) \to \mathsf{res}$: On input a functional key dk and a ciphertext ct, it outputs res as either the inner product $[\![\langle \boldsymbol{y}, \boldsymbol{x} \rangle]\!]_T$ or an error message $\perp$.

As shown, the secret key sk_i allows its owner to perform both encryption and functional-key generation. At a high level, the security of FH-IPFE ensures that given functional keys dk_i that encapsulate $[\![\boldsymbol{y}_i]\!]_1$ and ciphertexts ct_j that encapsulate $[\![\boldsymbol{x}_j]\!]_2$, no information beyond the inner products $\{[\![\langle \boldsymbol{y}_i, \boldsymbol{x}_j \rangle]\!]_T\}_{i,j}$ can be learned.

We also rely on a pseudorandom zero-sharing scheme [HRZ10, KDK11], in which each user locally generates a secret-public key pair and then non-interactively computes a pseudorandom share such that the sum of all users' shares equals zero. Assuming n users indexed by i, the scheme is instantiated using pairwise Diffie-Hellman key exchange over a cyclic group $\mathbb{G}_2$ as follows:

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$: On input a security parameter 1^λ, it samples a secret key $\mathsf{sk} = t$ and publishes the public key $[\![t]\!]_2$.

$\mathsf{KeyAgg}(i, \{\mathsf{pk}_k\}_{k \in [n] \setminus i}) \to \mathsf{pz}_i$: On input other public keys $\{[\![t_k]\!]_2\}_{k \in [n] \setminus i}$, the aggregate public key for computing the zero share of the user i is

$$\mathsf{pz}_i = \sum_{k \in [n] \setminus \{i\}} (-1)^{k<i} \cdot [\![t_k]\!]_2.$$

$\mathsf{ShareGen}(i, \mathsf{pz}_i, \mathsf{sk}_i) \to \mathsf{zero}_i$: On input the aggregate key pz_i and a secret key sk_i, it outputs $\mathsf{zero}_i = t_i \cdot \mathsf{pz}_i$ as the zero share of the user i.

From the above scheme, it holds that $\sum_{i\in[n]} \mathsf{zero}_i = [\![0]\!]_2$. Moreover, the aggregation procedure is deterministic and depends only on public keys. For security, the scheme provides one-time pseudorandomness, which is sufficient in the usual setting of a one-vote-per-voter election.

Warm-up protocol We assume a simplified setting with an authority (single trustee) that is responsible for registering voters and generating private encryption keys and a public decryption key for a random permutation in the offline phase. Assuming there are n registered voters, our protocol can be briefly described as follows:

- **Setup phase**: The authority generates, for each voter, a secret key sk_i of FH-IPFE for inner products of length three and a pseudorandom zero-sharing key t_i. The first two coordinates carry the vote and zero share, while the last coordinate is zero padding used in the security proof. Both keys are known only to the authority and the corresponding voter, while the public zero-sharing key $[\![t_i]\!]_2$ is published under voter i's identity.
  - *Permutation Generation*: The authority randomly generates a permutation Π over the index set $[n]$. Let $\boldsymbol{P}$ represent the corresponding permutation matrix for Π. In this matrix, each column $\boldsymbol{p}^i$ has an entry $p^i_j = 1$ if $j = \Pi(i)$, and $p^i_j = 0$ otherwise. This structure means that the vote of voter i will appear in position j in the result.
  - *Public-Key Aggregation*: The authority computes aggregate public keys of the zero-sharing scheme, namely pz_i for each voter i.
  - *Functional-Key Generation*: The authority generates a one-time pad vector $\boldsymbol{z}$, where each entry z_j corresponds to a row index j of the matrix $\boldsymbol{P}$. This one-time pad is used to re-randomise the zero shares in ciphertexts during FH-IPFE decryption. For each entry p^i_j, the authority generates a functional key dk^i_j for the vector $([\![p^i_j]\!]_1, [\![z_j]\!]_1, [\![0]\!]_1)$.

  The authority then publishes functional keys $\{\mathsf{dk}^i_j\}_{i,j\in[n]}$ that encode the hidden permutation Π, along with public encodings $[\![\boldsymbol{z}]\!]_1$ of the one-time pads and the aggregate public keys $\{\mathsf{pz}_i\}_{i\in[n]}$.
- **Voting phase**: Upon registration, each voter receives sk_i for FH-IPFE and t_i for zero-sharing. During the online voting phase, voter i uses the aggregate public key pz_i to compute its share zero_i, and encrypts the vector $([\![v_i]\!]_2, \mathsf{zero}_i, [\![0]\!]_2)$ into a ciphertext ct_i. The encrypted ballots are then submitted to a public ballot box.
- **Tallying phase**: After the voting phase concludes (with no additional ballots allowed), let $\mathsf{V} \subseteq [n]$ be the set of voters whose ballots are in the box. There are two cases:
  - If $\mathsf{V} = [n]$, i.e., all registered voters cast their ballots, the tally proceeds via decryption using the ciphertexts $\{\mathsf{ct}_i\}_{i\in[n]}$ and functional keys $\{\mathsf{dk}^i_j\}_{i,j\in[n]}$. The result is computed as:

$$\begin{aligned}
[\![\boldsymbol{v}']\!]_T &= \Big(\sum_{i\in[n]} \mathsf{Dec}(\mathsf{dk}^i_j, \mathsf{ct}_i)\Big)_{j\in[n]} \\
&= \Big(\sum_{i\in[n]} [\![p^i_j v_i]\!]_T + \sum_{i\in[n]} e([\![z_j]\!]_1, \mathsf{zero}_i)\Big)_{j\in[n]} \\
&= \Big(\sum_{i\in[n]} [\![p^i_j v_i]\!]_T\Big)_{j\in[n]} = [\![\boldsymbol{P}\boldsymbol{v}]\!]_T
\end{aligned}$$

    This result is thus a permuted version of the vote vector.
  - If $\mathsf{V} \subsetneq [n]$, i.e., some registered voters failed to submit their ballots, corrective action is required from the remaining honest voters. Each voter $i \in \mathsf{V}$ publishes a correcting key: $\mathsf{ck}_i = t_i \cdot (\sum_{k\in[n]\setminus\mathsf{V}} (-1)^{k<i} \cdot [\![t_k]\!]_2)$, which corresponds to the portion of randomness that was exchanged with dropout voters in zero_i. These corrections are used to cancel the residual terms introduced by the missing shares:

$$\begin{aligned}
[\![\boldsymbol{v}']\!]_T &= \Big(\sum_{i\in\mathsf{V}} \mathsf{Dec}(\mathsf{dk}^i_j, \mathsf{ct}_i) - \sum_{i\in\mathsf{V}} e([\![z_j]\!]_1, \mathsf{ck}_i)\Big)_{j\in[n]} \\
&= \Big(\sum_{i\in\mathsf{V}} [\![v_i p^i_j]\!]_T \\
&\qquad + \sum_{i\in\mathsf{V}} \big(e([\![z_j]\!]_1, \mathsf{zero}_i) - e([\![z_j]\!]_1, \mathsf{ck}_i)\big)\Big)_{j\in[n]} \\
&= [\![\boldsymbol{P}\boldsymbol{v}'']\!]_T.
\end{aligned}$$

In the final result, $v''_i = v_i$ for all $i \in \mathsf{V}$ and $v''_i = 0$ for all $i \in [n] \setminus \mathsf{V}$.

Our ballot privacy for this construction, under the selective and symmetric-key setting[5], holds against an unbounded number of corrupted voters. At a high level, the function-hiding property of each FH-IPFE instance ensures that both the vote v_i and the permutation image $\Pi(i)$, represented by the column vector $\boldsymbol{p}^i$ of the permutation $\boldsymbol{P}$, remain private. The decryption computes the inner product of each row j in $\boldsymbol{P}$ with the vote vector $\boldsymbol{v}$, i.e., $\sum_i v_i p^i_j$. At the same time, each re-randomised share $e([\![z_j]\!]_1, \mathsf{zero}_i)$ masks the partial decryption $[\![v_i p^i_j]\!]_T$.

Overall, this protocol can be viewed as a construction of multi-client functional encryption (MCFE) [GGG+14] for a hidden random permutation, which is secure against one functional key and one ciphertext vector under the assumption of a trusted key authority. To distribute trust away from this authority, we rely on the homomorphic properties of FH-IPFE to enable a distributed generation of the decryption key, as described next.

Voter registration via local key generation We now distribute the components generated in the setup phase, namely the FH-IPFE secret keys $\{\mathsf{sk}_i\}_{i\in[n]}$, the zero-sharing keys $\{t_i\}_{i\in[n]}$, the permutation $\boldsymbol{P}$, the one-time-pad vector $\boldsymbol{z}$, and the functional keys $\{\mathsf{dk}^i_j\}_{i,j\in[n]}$.

Since a self-tallying election typically involves a voter-registration phase, during which users authenticate themselves to become eligible voters, we can use this phase to let voters run the zero-sharing scheme in a decentralised manner by locally generating secret keys and publishing the corresponding public keys. The permutation and the one-time-pad vector can then be generated progressively by multiple trustees.

The main bottleneck, however, lies in the generation of the secret and functional keys, which embed the permutation and the one-time-pad vector. In particular, beyond relying on general-purpose multiparty computation protocols, there is no reasonably non-interactive and efficient approach for jointly generating these functional keys and privately issuing the corresponding secret keys to voters.

To circumvent this issue, our approach starts by letting each voter i generate sk_i and $\{\mathsf{dk}^i_j\}_{j\in[n]}$ as in the setup phase of the warm-up protocol, where $\boldsymbol{P}$ is initialised as the identity matrix. The mix-servers then shuffle and re-randomise these functional keys to embed their permutations. Although this keeps the distributed generation of the keys sk_i and dk^i_j relatively simple, the number of functional keys associated with each voter still grows linearly with the number of voters, resulting in an $O(n)$ complexity per voter.

To overcome this limitation, we combine key-homomorphic functional encryption with standard homomorphic public-key encryption (PKE) within the same pairing group. The key idea is to keep the generation of sk_i local to each voter, but rely on PKE to enable delegating the generation of $\{\mathsf{dk}^i_j\}_{j\in[n]}$ to the trustees, thereby reducing the voters' computational cost to $O(1)$. It then suffices to extend the setup of the warm-up protocol.

Setup phase now consists of 3 processes:

- **Election setup**: Through a standard distributed key generation, trustees generate a PKE public key such that each of them can provide a partial decryption.
- **Voter registration**: Each voter generates a pseudorandom zero-sharing key t_i and the following FH-IPFE keys.
  - It generates a secret key sk_i of FH-IPFE for inner products of length three, with a final zero-padding coordinate, and the following basis functional keys (whose subscripts indicate only the first two coordinates):
    1. $\mathsf{dk}^i_{(1,0)}$, corresponding to $[\![1,0,0]\!]_1$, for vote-slot extraction;
    2. $\mathsf{dk}^i_{(0,1)}$, corresponding to $[\![0,1,0]\!]_1$, for one-time-pad addition;
    3. $\mathsf{dk}^i_{(0,0)}$, corresponding to $[\![0,0,0]\!]_1$, for functional-key re-randomisation.
  - It encrypts the functional key for vote-slot extraction, namely $\mathsf{dk}^i_{(1,0)}$, under PKE to obtain ct_i. Importantly, the PKE layer ensures that the key cannot be used to decrypt individual ballots directly. Instead, after being processed by the trustees, it only appears as part of the final decryption key for the hidden permutation.

  The voter registers by publishing $\mathsf{pk}_i = (\mathsf{ct}_i, \mathsf{dk}^i_{(0,1)}, \mathsf{dk}^i_{(0,0)}, [\![t_i]\!]_2)$.
- **Public-key preprocessing**: Using the voter public keys, the trustees will jointly generate a decryption key for a permutation as follows.
  1. They parse the voter list as $\{\mathsf{pk}_i\}_{i\in[n]} = \{\mathsf{ct}_i, \mathsf{dk}^i_{(0,1)}, \mathsf{dk}^i_{(0,0)}, [\![t_i]\!]_2\}_{i\in[n]}$.
  2. For $i \in [n]$, they compute the aggregate public key pz_i from $\{[\![t_i]\!]_2\}_{i\in[n]}$.

[5] These settings are variants of our ballot privacy definition, given in Section 3.

3. The initial mix-box MBox_I is parsed by default as an $n \times n$-matrix where

$$\mathsf{MBox}_I(j,i) = \begin{cases} \mathsf{ct}_i & \text{if } j = i, \\ \mathsf{dk}^i_{(0,0)} & \text{if } j \neq i. \end{cases}$$

   Initially, this mix-box corresponds to the identity permutation.
4. OMIX protocol: The k-th trustee generates
   - a random permutation $\Pi_k : [n] \to [n]$;
   - PKE randomness $(\gamma_{\mathsf{PKE},k,j,i})_{j,i\in[n]}$;

   and mixes MBox_{k-1} into MBox_k (where $\mathsf{MBox}_0(j,i) = \mathsf{MBox}_I(j,i)$) by permuting and rerandomizing each entry as

$$\mathsf{MBox}_k(\Pi_k(j),i) = \mathsf{MBox}_{k-1}(j,i) + \gamma_{\mathsf{PKE},k,j,i} \cdot \mathsf{ct}_0.$$

   where ct_0 is some ciphertext randomiser of PKE.
5. OPAD protocol: The k-th trustee generates
   - randomness $\{\gamma_{\mathsf{IPE},k,j,i}\}_{j,i\in[n]}$ for functional keys;
   - a one-time-pad vector $\boldsymbol{z}_k = (z_{k,j})_{j\in[n]}$.

   It encrypts each one-time-pad entry as $\mathsf{ct}^{\mathsf{PAD}}_{k,j}$. It generates OBox_k as an $n \times n$ matrix where each entry

$$\mathsf{OBox}_k(j,i) \text{ is a PKE ciphertext of } (z_{k,j} \cdot \mathsf{dk}^i_{(0,1)} + \gamma_{\mathsf{IPE},k,j,i} \cdot \mathsf{dk}^i_{(0,0)}).$$

6. Trustees jointly decrypt the sum $\sum_k \mathsf{ct}^{\mathsf{PAD}}_{k,j}$ to reveal $[\![\sum_k z_{k,j}]\!]_1$. Parsing $\boldsymbol{z} = \sum_k \boldsymbol{z}_k$, they obtain $[\![\boldsymbol{z}]\!]_1$.
7. Trustees obtain the functional keys $\{\mathsf{dk}^i_j\}_{i,j\in[n]}$ by jointly decrypting the sums

$$\sum_k \mathsf{OBox}_k(j,i) + \mathsf{MBox}(j,i).$$

The preprocessing phase outputs $\{\mathsf{pz}_i\}_{i\in[n]}$, $[\![\boldsymbol{z}]\!]_1$ and $\{\mathsf{dk}^i_j\}_{i,j\in[n]}$.

Under the above setup extension, the voting and tallying phases remain unchanged from the warm-up protocol. For the distributed key generation, as long as at least one trustee is honest and others are honest-but-curious, the distributions of the preprocessed components $\{\mathsf{pz}_i\}_{i\in[n]}$, $[\![\boldsymbol{z}]\!]_1$, and $\{\mathsf{dk}^i_j\}_{i,j\in[n]}$ remain unchanged. In particular, the permutation embedded in $\{\mathsf{dk}^i_j\}_{i,j\in[n]}$ corresponds to the composition of the trustees' permutations Π_k, thereby preserving voter privacy.

On the other hand, the protocol OMIX is the only component of the preprocessing phase that incurs non-linear operations and lacks a dedicated verification mechanism. In the following, based on an enhanced version of linearly homomorphic signatures, we describe how to ensure security in the presence of malicious mixing trustees.

Verifiable OMIX from enhanced randomisable-tag LHS For the verifiability of the OMIX protocol, the main idea is to use an LHS scheme to sign the ciphertexts $\mathsf{ct}_{j,i}$ of the initial mix-box MBOX_I, together with the ciphertext randomiser ct_0, under two-dimensional tags, where one dimension is bound to the row index j and the other to the column index i.

Since re-randomisation of a ciphertext $\mathsf{ct}_{i,j}$ consists of adding to it a multiple of ct_0, the linearly homomorphic property of LHS ensures that the resulting ciphertext remains validly signed. Conversely, unforgeability guarantees that any transformation other than valid re-randomisation yields an invalid signature.

As mix-servers permute the rows of ciphertexts during the shuffle, the row tags must themselves be re-randomisable in order to preserve unlinkability of the applied permutation. However, this feature must be incorporated carefully to ensure that a malicious mix-server cannot combine ciphertexts originating from different rows by rerandomizing their row tags into a common one.

Constructing an LHS scheme satisfying these requirements is non-trivial. In particular, existing definitions of linearly homomorphic signatures do not guarantee that all elements in a re-randomised row vector come from the same initial row. This is because standard unforgeability only applies to each *individual* signature under a re-randomised tag, but does not guarantee consistency across an entire row vector.

To address this issue, we strengthen the standard definition of unforgeability to a notion called *tag-binding* unforgeability, which guarantees that all elements signed under a given output row tag are generated

from those under a *unique* initial row tag. Whilst it is clear that tag-binding unforgeability implies standard unforgeability, we complete this technical contribution by showing the non-trivial direction, proving that any LHS scheme with a property called *homomorphic verification* satisfies tag-binding unforgeability.

As a second contribution to LHS, we incorporate two-dimensional tags into the tag-binding unforgeable LHS framework, where row tags remain rerandomisable while still binding ciphertexts to their matrix positions. To achieve this, we extend the generic conversion from one-time-LHS to LHS from [HPP20] to the signatures on equivalence classes from [FHS19], yielding an LHS scheme that satisfies the requirements of OMIX.

Final Construction By instantiating the protocol with a verifiable OMIX, we obtain a mix-net-based voting scheme that simultaneously achieves self-tallying, corrective fault tolerance, client-side lightweightness, universal verifiability, and ballot privacy in a multi-trustee setting.

Scope and open directions Our analysis establishes ballot privacy in the selective, symmetric-key setting, with corrupted preprocessing servers fixed in advance and an honest contribution in each of the mixing and padding protocols. The offline mix-net uses honest input certification. Extending these guarantees to adaptive corruption, including later exposure of trustee randomness, and distributing certification are natural directions for further work; neither extension follows automatically from secure erasure or an honest mixing contribution.

For shuffle-based tallying, privacy protects the association between voters and the revealed plaintext multiset, subject to information inferable from that output. Correcting keys and participation in correction are public and identity-linked. Receipt-freeness and coercion resistance would strengthen this privacy goal and remain open in our framework. Likewise, recovery assumes that all participating voters supply valid correcting keys after successful preprocessing. Handling withheld correction keys and preprocessing aborts, including restart and safe reuse, calls for an additional recovery layer and analysis. Finally, reducing the quadratic helper-key size, preprocessing, and tallying costs is an important step towards system-wide scalability. These questions build on the separation between offline trustee work and lightweight online voting developed here.

1.3 Related Work

Electronic voting is a dynamic research domain where numerous innovative cryptographic tools have been applied to strike a balance between usability, privacy, verifiability, and efficiency. We highlight significant technical advances that are related to our work.

Ballot-private Voting Electronic voting systems can generally be categorised based on two key tallying methods: homomorphic aggregation such as [Adi08, CGG19, AW07b, HMMP23, Poi24] and mix-net-based shuffle such as [SK95, JCJ05, Ben06, CCM08, Wik18, The18, HMMP23]. In this work, although we focus on mix-net-based voting, our formalisations capture other tallying methods in the self-tallying setting.

For ballot privacy, the search for security models began with the work of Benaloh [CF85, Ben87] and has since evolved into a well-established game-based definition in [BCG$^+$15]. This latter security game effectively addresses privacy concerns, such as protection against replay attacks [CS11], while still allowing important features like verifiability and capturing a variety of result functions.

In this work, we adopt the definition of Benaloh in [Ben87] to capture ballot privacy for our voting protocols, rather than relying on recent definitions in [BCG$^+$15, CCFG16, CFL19]. This choice is motivated by the nature of the self-tallying setting, where decryption is public and thus cannot be simulated. As a result, an admissibility condition on vote queries, which was originally introduced in [Ben87], is necessary. As a contribution, we propose a slight modification to Benaloh's definition to mitigate its known limitation on the class of allowable result functions, as identified in [BCG$^+$15].

Self-tallying Elections The concept of self-tallying voting was first introduced by Kiayias and Yung in [KY02], allowing any party, including voters and external observers, to compute the election outcome once all ballots have been cast. This eliminates the need for trusted talliers to compute the result.

This line of research has seen various developments aimed at improving efficiency and security [Gro04a, HRZ10, KSRH12, MSH17, LSY$^+$22, EY23, EYH25] and supporting more expressive ballot formats [Gro04a, AAG$^+$25, YGW$^+$21, PBHR20, ZZH$^+$25]. Most existing works also focus on the so-called maximal ballot secrecy setting, where ballot privacy is decentralised among the voters themselves. This typically requires each voter to audit the list of participants with whom they wish to vote, resulting in per-voter costs that grow with the total number of participants. Consequently, such systems are generally suitable

only for restricted-scale elections. This motivates the question of how to leverage servers' preprocessing power to offload the voters' heavy computational tasks, thereby enabling self-tallying elections at a larger scale.

Verifiable and Offline Mix-nets Research on mix-nets is a highly active field of study, with the primary research focus being the challenge of developing more efficient protocols for deployment in real-world scenarios such as e-voting and anonymous routing. A vast number of works have aimed to reduce the concrete computational and communication overheads, for example [BG12, FS01, GI08, HPP20, HM20, Wik09, TW10, FLSZ17, FR22, ANO+25]. Despite these advancements, the overall cost of mixing remains linear in the number of mixing rounds and requires a global verification of all ciphertexts.

To address the above issues, another direction is to use the Certified-Input Mixnets (CIM) model [HPP20, ANO+25], where the common reference string (CRS) is generated depending on each input ciphertext. Concretely, each ciphertext is signed prior to mixing and each signature serves as an independent CRS to verify the re-randomisation of its corresponding ciphertext. This approach helps remove the dependency on the number of mix-servers in both proof size and online verification time, and simplifies the proof to per-ciphertext verification. Therefore, OMIX can be modelled as an *offline* CIM, using an enhanced version of linearly homomorphic signatures.

Notably, a separate line of research [AW07a, AW07b, PRT12] explores offline mix-nets, where the mixing process is carried out entirely during a pre-computation phase. This design shifts the overhead of sequential mixing away from the tallying phase, enabling plausibly parallelisable result computation. However, these protocols do not achieve self-tallying, as they critically rely on a set of decryption trustees to produce the final outcome.

Multi-Client Functional Encryption for Selection The work by Shi and Wu in [SW21] introduced a primitive called multi-client functional encryption for selection, which allows multiple independent clients to send their encrypted inputs, such that a functional decryption key given by a trusted authority will secretly embed a selection over the input entries and return only the selected entry when decrypting client ciphertexts. This primitive is used as a building block to construct a non-interactive anonymous router (NIAR) when the permutation is predefined and a non-interactive anonymous shuffler (NIAS) when the permutation is random.

Building on this connection between function hiding and hidden selection, our warm-up protocol uses pairing-based FH-IPFE, following the single-client-to-multi-client perspective of [AGT21], to explain how public functional keys can reveal a shuffled vote vector. Our contribution centres on preparing those keys distributively and certifying their preparation before voting, which introduces the following requirements:

- **Distributed permutation preparation.** The trusted setup in Shi and Wu's construction already embeds the hidden selection into functional keys. In our setting, multiple trustees must contribute to a hidden permutation of independently registered voters' keys. We realise this through a row shuffle of an encrypted functional-key matrix, together with a separate padding protocol. A hidden-selection FE interface alone does not specify or certify these transformations.
- **Certification of matrix transformations.** Correctness requires entries of an output row to originate from one input row, while retaining their column bindings. Ordinary per-entry LHS unforgeability does not by itself express this cross-entry requirement. Tag-binding addresses common row provenance, and two-dimensional tags combine rerandomisable row tags with fixed column bindings. These are the additional properties used by our verifiable offline shuffle, rather than treating a vector shuffle proof as a black-box matrix shuffle proof.
- **One-time voting and client costs.** Our algebraic zero shares support aggregation over public keys in the single-election setting. This permits voter-side registration and casting costs independent of the number of voters, while moving quadratic work to preprocessing and tallying. This is a setting-specific trade-off, not a general efficiency improvement over MCFE for selection.

These distinctions identify the requirements addressed by our construction, while leaving room for alternative approaches based on MCFE for selection. Distributed permutation preparation and honest input certification play distinct roles in our certified-input model.

1.4 Organisation of the Paper

The paper develops the voting architecture from its security goals to a concrete distributed instantiation. We first establish the notation and standard cryptographic primitives in Section 2. We then specify the

voting interface and the guarantees to be achieved in Section 3: correctness, corrective fault tolerance, client-side lightweightness, ballot privacy, and universal verifiability, followed by a discussion of fairness and dispute-freeness.

With these goals in place, Section 4 introduces a single-trustee warm-up protocol to explain how function-hiding inner-product encryption enables public shuffled tallying, and proves its privacy. Building on this mechanism, Section 5 presents the voting framework with locally generated voter keys and offline mixing and padding, together with its correctness and security proofs. To instantiate the mixing component, Section 6 develops the tag-binding and two-dimensional-tag LHS tools needed to certify matrix row shuffles. Finally, Section 7 formalises and analyses the offline protocols, constructs the distributed padding component, and establishes their secure integration into the voting framework. This completes the progression from the public-tallying mechanism to certified distributed preprocessing.

2 Notation and Standard Primitives

We introduce the notation, group assumptions, public-key encryption, and proof systems used below. Voting definitions follow in Section 3; functional encryption and linearly homomorphic signatures are introduced alongside their applications in Sections 4.1 and 6.1, respectively.

2.1 Notation

Sets. Given any $n \in \mathbb{N}$, we denote by $[n]$ the set of integers $\{1, \ldots, n\}$. We denote by $|\mathcal{S}|$ the cardinality of any finite set $\mathcal{S}$. For any $p \in \mathbb{N}$, $\mathbb{Z}_p$ typically denotes the integers modulo p. For any vector $\boldsymbol{a} \in \mathbb{Z}_p^n$, we use a subscript i to denote by a_i the i-th component of $\boldsymbol{a}$. Similarly, for any matrix $\boldsymbol{A} \in \mathbb{Z}_p^{m \times n}$, we use subscripts (i, j) to denote by $A_{i,j}$ the component in the i-th row and the j-th column of $\boldsymbol{A}$. The identity matrix is denoted by $\boldsymbol{I}$.

Permutations. Let $\Pi : [n] \to [n]$ be a bijective map. For every $\boldsymbol{a} \in \mathbb{Z}_p^n$, we let $\mathsf{PerVec}_\Pi(\boldsymbol{a})$ be the vector generated by permuting n entries of $\boldsymbol{a}$ with respect to Π, i.e. if $\boldsymbol{a'} = \mathsf{PerVec}_\Pi(\boldsymbol{a})$ then $a'_{\Pi(i)} = a_i$ for all $i \in [n]$. We also denote by $\mathsf{PerMatRow}_\Pi(\boldsymbol{I})$ the matrix generated by permuting n rows of an identity matrix $\boldsymbol{I}$ with respect to Π so that $\boldsymbol{a'} = \mathsf{PerMatRow}_\Pi(\boldsymbol{I})\boldsymbol{a}$. Columns of $\mathsf{PerMatRow}_\Pi(\boldsymbol{I})$ can be parsed as $(\boldsymbol{p}^i)_{i \in [n]}$ where $p^i_j = 1$ if $j = \Pi(i)$ and $p^i_j = 0$ otherwise.

2.2 Groups

Let GGen be a prime-order group generator, a probabilistic polynomial time (PPT) algorithm that on input the security parameter 1^λ returns a description $\mathcal{G} = (\mathbb{G}, p, P)$ of an additive cyclic group $\mathbb{G}$ of prime order p, where p is a 2λ-bit prime and P is its generator:

- for $a \in \mathbb{Z}_p$, we define $[\![a]\!] = a \cdot P \in \mathbb{G}$ as the *implicit representation* of a in $\mathbb{G}$;
- field multiplication in $\mathbb{Z}_p$ is written by juxtaposition, as ab. Scalar multiplication in $\mathbb{G}$ is written $a \cdot X$ for $a \in \mathbb{Z}_p$ and $X \in \mathbb{G}$, with the scalar on the left. This notation extends componentwise to vectors of group elements, including ciphertexts and functional keys. Addition is written $+$ in both $\mathbb{Z}_p$ and $\mathbb{G}$; matrix products and scalar multiples of vectors over $\mathbb{Z}_p$ are written by juxtaposition;
- inner products between vectors $\boldsymbol{a} \in \mathbb{Z}_p^\rho$ and vectors $\boldsymbol{b} \in \mathbb{Z}_p^\rho$ (or between $\boldsymbol{a} \in \mathbb{Z}_p^\rho$ and $[\![\boldsymbol{b}]\!] \in \mathbb{G}^\rho$) are denoted by $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ (or $\langle \boldsymbol{a}, [\![\boldsymbol{b}]\!] \rangle$, respectively).

Given $[\![a]\!], [\![b]\!] \in \mathbb{G}$ and a scalar $x \in \mathbb{Z}_p$, one can efficiently compute $[\![ax]\!] = x \cdot [\![a]\!] \in \mathbb{G}$ and $[\![a + b]\!] = [\![a]\!] + [\![b]\!] \in \mathbb{G}$. We rely on the following standard assumptions.

Definition 1 (Discrete Logarithm (DL) Assumption). *In a group $\mathbb{G}$ of prime order p, it states that for any random element $[\![x]\!]$, it is computationally hard to recover x.*

Definition 2 (Decisional Diffie-Hellman (DDH) Assumption). *In a group $\mathbb{G}$ of prime order p, it states that for any random elements $[\![x]\!]$ and $[\![y]\!]$, it is computationally hard to distinguish between $[\![xy]\!]$ and a random element $[\![z]\!]$.*

Definition 3 (n-Multi-DDH Assumption). *In a group $\mathbb{G}$ of prime order p, it states that for any random elements $[\![x]\!]$ and $\{[\![y_i]\!]\}_{i \in [n]}$, it is computationally hard to distinguish between $\{[\![xy_i]\!]\}_{i \in [n]}$ and random $\{[\![z_i]\!]\}_{i \in [n]}$.*

We refer to [CDG+18] for a self-reduction from DDH to n-Multi-DDH.

2.3 Asymmetric Pairing Groups

Let PGGen be a pairing group generator, a PPT algorithm that on input the security parameter 1^λ returns a description $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, P_1, P_2, e)$ of asymmetric pairing groups in which $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are additive cyclic groups of order p, where p is a 2λ-bit prime, P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map. Let $P_T := e(P_1, P_2)$. For $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$, we define $[\![a]\!]_s = a \cdot P_s \in \mathbb{G}_s$ as the implicit representation of a in $\mathbb{G}_s$.

Pairings are always written explicitly as $e(X, Y)$ with $X \in \mathbb{G}_1$ and $Y \in \mathbb{G}_2$; in particular,

$$e([\![a]\!]_1, [\![b]\!]_2) = [\![ab]\!]_T, \qquad a \cdot [\![b]\!]_s = [\![ab]\!]_s.$$

Thus, $\cdot$ never denotes a pairing or a product of two group elements. The DDH assumption can be extended to $\mathcal{PG}$ as follows.

Definition 4 (Symmetric eXternal Diffie-Hellman (SXDH) Assumption). *In a pairing group $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, P_1, P_2, e)$ of prime order p, it states that the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

2.4 Public Key Encryption

We recall the notion of public key encryption, along with relevant properties.

Definition 5 (PKE). *A public-key encryption scheme over a group $\mathcal{G} = (\mathbb{G}, p)$ consists of three algorithms:*

- $\mathsf{SetUp}(1^\lambda, 1^\rho)$*: On input a security parameter λ and a dimension ρ, it outputs a public/secret key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, [\![\boldsymbol{m}]\!])$*: On input a public key* pk *and a message vector $[\![\boldsymbol{m}]\!] \in \mathbb{G}^\rho$, it outputs a ciphertext* ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: On input a secret key* sk *and a ciphertext* ct*, it outputs an element in $\mathbb{G}^\rho$.*

Correctness For all parameters $\lambda, \rho \in \mathbb{N}$, key pairs $(\mathsf{pk}, \mathsf{sk})$ issued by $\mathsf{SetUp}(1^\lambda, 1^\rho)$, and messages $[\![\boldsymbol{m}]\!] \in \mathbb{G}^\rho$, if $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, [\![\boldsymbol{m}]\!])$, then $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = [\![\boldsymbol{m}]\!]$.

Definition 6 (Security). *Let* PKE *be a public-key encryption scheme. We define the experiment* $\mathrm{IND}_b^{\mathsf{PKE}}$ *in Figure 5, where $b \in \{0, 1\}$ and the oracles are defined as:*

$\underline{\mathrm{IND}_b^{\mathsf{PKE}}(\lambda, \rho, \mathcal{A})}$:
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SetUp}(1^\lambda, 1^\rho)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{QEnc}_b(\cdot,\cdot)}(\mathsf{pk})$
Output: $\beta = 1$ if $\alpha = b$, and $\beta = 0$ otherwise.

Fig. 5: Security experiment for PKE

- ***Encryption oracle*** $\mathsf{QEnc}_b([\![m^0]\!], [\![m^1]\!])$*: it returns* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, [\![m^b]\!])$.

The PKE *scheme is said to be* IND-CPA*-secure if, for any parameter $\rho \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$, the following advantage is negligible in λ:*

$$\mathsf{Adv}^{\mathrm{IND}^{\mathsf{PKE}}}(\lambda, \rho, \mathcal{A}) := \left| \Pr[\mathrm{IND}_0^{\mathsf{PKE}}(\lambda, \rho, \mathcal{A}) = 1] - \Pr[\mathrm{IND}_1^{\mathsf{PKE}}(\lambda, \rho, \mathcal{A}) = 0] \right|.$$

Since the experiment outputs 1 exactly when $\alpha = b$, the two probabilities above are respectively $\Pr[\alpha = 0 \mid b = 0]$ and $\Pr[\alpha = 0 \mid b = 1]$. Thus, this convention is equivalent to comparing the same output bit of the adversary in the two worlds.

Definition 7 (Homomorphic Ciphertext). *A* PKE *scheme supports homomorphic ciphertexts if, given any two encryptions $\mathsf{ct}_1 = \mathsf{Enc}(\mathsf{pk}, [\![\boldsymbol{m}_1]\!]; r_1)$ and $\mathsf{ct}_2 = \mathsf{Enc}(\mathsf{pk}, [\![\boldsymbol{m}_2]\!]; r_2)$ (in the form of vectors in $\mathbb{G}$) under randomness r_1 and r_2 (in the form of elements in $\mathbb{Z}_p$) respectively, for any scalars $\alpha_1, \alpha_2 \in \mathbb{Z}_p$, the following holds:*

$$\alpha_1 \cdot \mathsf{ct}_1 + \alpha_2 \cdot \mathsf{ct}_2 = \mathsf{Enc}(\mathsf{pk}, [\![\alpha_1 \boldsymbol{m}_1 + \alpha_2 \boldsymbol{m}_2]\!]; \alpha_1 r_1 + \alpha_2 r_2)$$

2.5 Non-Interactive Zero-Knowledge Proofs

Let $R(\cdot;\cdot)$ be a relation that is computable in polynomial time with respect to the size of its first input. Associated with R, we define the NP language $L_R = \{x \mid \exists w \text{ such that } R(x;w) = \mathsf{true}\}$. A value w is called a *witness* for the statement x if $R(x;w) = \mathsf{true}$, which we denote as $(x;w) \in R$.

Definition 8 (NIZK). *A non-interactive proof system for a relation R with setup consists of three algorithms:*

- $\mathsf{SetUp}(1^\lambda)$*: On input a security parameter λ, it outputs a common reference string σ;*
- $\mathsf{Prove}(\sigma, x; w)$*: On input a common reference string σ and a pair of statement and witness $(x;w)$, it outputs a proof π;*
- $\mathsf{Verify}(\sigma, x, \pi)$*: On input a common reference string σ and a pair of statement and proof (x,π), it outputs 1 to accept and 0 to reject;*

These algorithms satisfy the following properties:

- *Completeness. For all $\sigma \leftarrow \mathsf{SetUp}(1^\lambda)$ and $(x;w) \in R$, the probability that $\mathsf{Verify}(\sigma, x, \pi) = 1$ for $\pi \leftarrow \mathsf{Prove}(\sigma, x; w)$ is 1.*
- *Soundness. For every PPT adversary $\mathcal{A}$ and $\sigma \leftarrow \mathsf{SetUp}(1^\lambda)$, the probability that $\mathcal{A}(\sigma)$ outputs (x,π) such that $x \notin L_R$ but $\mathsf{Verify}(\sigma, x, \pi) = 1$ is negligible.*
- *Zero-Knowledge. There exists a polynomial-time simulator algorithm $S = (S_1, S_2)$ such that $S_1(1^\lambda)$ outputs (σ_{sim}, τ_s) and $S_2(\sigma_{sim}, \tau_s, x)$ outputs a value π_s such that for all $(x;w) \in R$ and PPT adversaries $\mathcal{A}$, the following two interactions are indistinguishable: in the first, we compute $\sigma \leftarrow \mathsf{SetUp}(1^\lambda)$ and give σ to $\mathcal{A}$ and oracle access to $\mathsf{Prove}(\sigma, \cdot;\cdot)$ (where Prove outputs $\perp$ on input $(x;w)$ such that $(x;w) \notin R$); in the second, we compute (σ_{sim}, τ_s) and give σ_{sim} to $\mathcal{A}$ and oracle access to $S(\sigma_{sim}, \tau_s, \cdot, \cdot)$ where, on input $(x;w)$, S outputs $S_2(\sigma_{sim}, \tau_s, x)$ if $(x;w) \in R$ and $\perp$ otherwise. Perfect zero-knowledge is achieved if for all $(x;w) \in R$, these interactions are distributed identically.*

When the common reference string and the statement are clear from the context, we assume that they are contained in the proof.

(Non-interactive) Σ-protocols. A Σ-protocol is a special type of 3-move proof system. Given a common reference string σ containing a relation R, a statement x, and a witness w such that $R(x;w) = \mathsf{true}$, the protocol proceeds as follows: the prover sends an initial message a, receives a challenge e, and produces an answer z. The verifier can then evaluate (σ, x, a, e, z) and decide whether to accept or reject the proof.

Given access to an oracle $\mathcal{RO}$, we can transform a Σ-protocol into a non-interactive proof system. To obtain the challenge e, we form the initial message a, query $\mathcal{RO}$ on (x, a, aux)[6] to obtain the challenge e, and then compute the answer z. The proof is then $\pi = (a, z, \mathsf{aux})$. To verify such a proof, one queries $\mathcal{RO}$ on (x, a, aux) to obtain e and then runs the verifier from the Σ-protocol.

The random oracle model is an idealization of the Fiat-Shamir heuristic [BR93]. In practice, a prover uses a cryptographic hash function $\mathcal{H}$ to produce the challenge as $e = \mathcal{H}(x, a, \mathsf{aux})$.

3 Voting Definitions

3.1 Voting Syntax

To formally present our approach, we introduce a voting model that supports private-key voting, fault correction, and self-tallying. While the notion of a private voting key is present in earlier works, it typically refers to the secret keys associated with eligibility credentials. In our model, the private voting key will additionally contain a private encryption key to ensure ballot privacy[7]. Furthermore, as shown in Proposition 1 of [KY02], no self-tallying voting scheme can achieve robustness and privacy simultaneously. Therefore, fault tolerance should be realised through corrective actions taken by either honest voters or authorities.

In practice, voting systems usually involve multiple authorities that are responsible for various tasks. We specify the following entities:

[6] As in [Gro04b], the auxiliary information aux will contain the identity of the prover (the voter in this work) to prevent somebody else duplicating the proof without knowing the witness.

[7] Throughout this work, we implicitly assume the existence of an authentication mechanism for voter eligibility.

1. *Election administrator*, denoted by $\mathcal{E}$, sets up the election by publishing the universe $\mathcal{U}$ of voters, the list of candidates and the result function F of the election.
2. *(Public) Registrar*, denoted by $\mathcal{R}$, issues voting eligibility to voters based on their public keys. Everybody should be able to check the validity of each public key. The registrar publishes and updates the list of eligible voters from the universe $\mathcal{U}$, along with their public keys.
3. *Trustee*, denoted by $\mathcal{T}$, generates master secret/public keys for the election (possibly in a thresholdized manner) and processes the public keys of eligible voters to generate a *helper public key*. In our context, trustees include servers that execute mixing and decryption.
4. A set of *Voters*, identified by id in a universe $\mathcal{U}$, who each generate and send their public key to $\mathcal{R}$ to register for voting eligibility. Each voter generates and stores their private encryption key during the computation of the public key.
5. *(Public) Ballot-box Manager*, denoted by $\mathcal{B}$, processes and stores valid ballots in the public ballot box BB. By handling ballot submission, it also manages the list of dropout voters.
6. *(Public) Tallier*, denoted by $\mathcal{D}$, computes the final result based on BB and possibly auxiliary public components such as correcting keys.

In the following, we provide a definition for single-vote systems, in which each voter can generate at most one ballot.

Definition 9 (Voting System). *A voting scheme $\mathcal{V}ot$ for a universe of voters $\mathcal{U}$, a vote space $\mathbb{V}$ and a result function $F : (\mathcal{U} \times \mathbb{V})^* \to \{0,1\}^*$ consists of the following algorithms.*

- $\mathsf{SetUp}(1^\lambda)$ *takes as input a security parameter 1^λ. The list of eligible voters L and the public ballot box BB are initialised to be empty. It outputs a master public/secret key pair $(\mathsf{mpk}, \mathsf{msk})$. The master public key mpk is an implicit argument to other algorithms.*
- $\mathsf{KeyGen}(\mathsf{id}, \mathsf{msk})$ *is used by an authenticated voter $\mathsf{id} \in \mathcal{U}$ to generate a private encryption key $\mathsf{ek}_{\mathsf{id}}$ (possibly with the trustee when msk is needed as input) and a voter public key $\mathsf{pk}_{\mathsf{id}}$. It outputs $(\mathsf{pk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}})$. The correspondence between id and $\mathsf{pk}_{\mathsf{id}}$ is public.*
- $\mathsf{Register}(\mathsf{L}, \mathsf{pk}_{\mathsf{id}})$ *updates L with the public key $\mathsf{pk}_{\mathsf{id}}$. Typically, this consists of adding $\mathsf{pk}_{\mathsf{id}}$ as a new entry to L, depending on the validity of $\mathsf{pk}_{\mathsf{id}}$. It also outputs 1 if $\mathsf{pk}_{\mathsf{id}}$ is successfully appended, and 0 otherwise.*
- $\mathsf{PKProcess}(\mathsf{L}, \mathsf{msk})$ *takes as input a list of eligible public keys L and the master secret key msk. It outputs a helper public key hpk, which possibly contains mpk.*
- $\mathsf{Vote}(\mathsf{ek}_{\mathsf{id}}, v, \mathsf{hpk})$ *is run by a voter id with the private encryption key $\mathsf{ek}_{\mathsf{id}}$, a vote $v \in \mathbb{V}$, and the helper public key hpk. It outputs a ballot b_{id}. We assume that the correspondence between id and b_{id} is public and that $v = \perp$ means no vote is sent*[8]*.*
- $\mathsf{Append}(\mathsf{BB}, b_{\mathsf{id}}, \mathsf{L}, \mathsf{hpk})$ *updates BB with the ballot b_{id} and the list of eligible voters L. Typically, this consists of adding b_{id} as a new entry to BB, depending on the validity of b_{id} and the eligibility of id. It also outputs 1 if b_{id} is successfully appended and 0 otherwise (e.g., if it is malformed, submitted by a voter who has already voted, or submitted by a non-eligible voter).*
- $\mathsf{FaultAgg}(\mathsf{BB}, \mathsf{hpk}, \mathsf{id})$ *takes as input the ballot box BB, the helper public key hpk and the identity id. It outputs a fault aggregate $\mathsf{fa}_{\mathsf{id}}$ that might contain information about dropout voters for fault correction by voter id.*
- $\mathsf{FaultCrt}(\mathsf{fa}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}})$ *takes as input the fault aggregate $\mathsf{fa}_{\mathsf{id}}$ and the private encryption key $\mathsf{ek}_{\mathsf{id}}$. It outputs a correcting key $\mathsf{ck}_{\mathsf{id}}$ for fault correction.*
- $\mathsf{SelfTally}(\mathsf{BB}, \{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}, \mathsf{hpk})$ *takes as input the ballot box BB, possibly the correcting keys $\{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}$, and the helper public key hpk. It outputs a result res (possibly $\mathsf{res} = \perp$).*
- $\mathsf{Verify}(\mathsf{BB}, \{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}, \mathsf{res}, \mathsf{L}, \mathsf{hpk})$ *takes as input the ballot box BB, the correcting keys $\{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}$, a result res, a list L of eligible voters and the helper public key hpk. It returns 1 if valid, and 0 otherwise.*

The offline phase consists of four algorithms. SetUp is run by the election administrator and trustee to generate system parameters and a master public key, respectively. KeyGen is run by each authenticated voter, optionally with trustee interaction, to generate a private encryption key and submit a corresponding public key to the registrar. $\mathsf{Register}$, run by the registrar, grants voting eligibility and compiles the list of eligible voters. Finally, $\mathsf{PKProcess}$ is run by the trustee on all registered public keys to produce a helper public key, which serves as a reference string and encapsulates the necessary decryption keys for shuffling.

When no fault correction is needed, the correcting-key input to $\mathsf{SelfTally}$ and Verify is the empty set $\emptyset$.

The online phase contains six algorithms. Vote is run by each voter to cast their ballot. Append, run by the ballot-box manager, ensures only valid ballots are recorded. $\mathsf{FaultAgg}$ is a publicly computable

[8] The null value $\perp$ is not contained in the valid vote space $\mathbb{V}$.

procedure, run by the ballot-box manager. $\mathsf{FaultCrt}$ is run by each of the remaining voters after the ballot-casting phase to handle faults. $\mathsf{SelfTally}$ enables anyone to compute the election result without secret information, and Verify provides public verification of the result and the associated protocol data.

Correctness ensures that when all parties run the voting protocol honestly, the final result correctly reflects the intended votes and all verifications are valid. Correctness is defined as follows:

Definition 10 (Correctness). *A voting scheme $\mathcal{V}ot$ is said to be correct if, for all parameters λ and all integers n, the following properties hold with overwhelming probability in λ: given $(\mathsf{mpk}, \mathsf{msk})$ issued by $\mathsf{SetUp}(1^\lambda)$, $(\mathsf{pk}_i, \mathsf{ek}_i)$ issued by $\mathsf{KeyGen}(\mathsf{id}_i, \mathsf{msk}, \mathsf{mpk})$ for $i \in [n]$, $\mathsf{L}_i := (\mathsf{pk}_1, ..., \mathsf{pk}_i)$ for all $i \in [n]$ and $L_0 = \emptyset$, hpk issued by $\mathsf{PKProcess}(\mathsf{L}_n, \mathsf{msk})$, votes $(v_i)_{i\in[n]} \in \mathbb{V}^n$ from voters $(\mathsf{id}_i)_{i\in[n]}$, ballots b_i issued by $\mathsf{Vote}(\mathsf{ek}_i, v_i, \mathsf{hpk})$, and $\mathsf{BB}_i := (b_1, ..., b_i)$ for all $i \in [n]$ and $\mathsf{BB}_0 = \emptyset$, then*

- *public keys are registered successfully, that is, $\mathsf{Register}(\mathsf{L}_{i-1}, \mathsf{pk}_i) = (\mathsf{L}_i, 1)$ for all $i \in [n]$;*
- *ballots are appended successfully, that is, $\mathsf{Append}(\mathsf{BB}_{i-1}, b_i, \mathsf{L}_n, \mathsf{hpk}) = (\mathsf{BB}_i, 1)$ for all $i \in [n]$;*
- *the tally is correct and valid, that is, given $\mathsf{res} = \mathsf{SelfTally}(\mathsf{BB}_n, \emptyset, \mathsf{hpk})$ then*

$$\mathsf{res} = F((\mathsf{id}_i, v_i)_{i\in[n]}) \text{ and } \mathsf{Verify}(\mathsf{BB}_n, \emptyset, \mathsf{res}, \mathsf{L}_n, \mathsf{hpk}) = 1.$$

To define the functionality for mix-net-based voting, we define an equivalence relation $\sim_n$ on a space $\mathbb{V}^n$ for any integer n as follows: given any tuples $\boldsymbol{v} = (v_i)_{i\in[n]}, \boldsymbol{v}' = (v'_i)_{i\in[n]} \in \mathbb{V}^n$, we have $\boldsymbol{v} \sim_n \boldsymbol{v}'$ if there exists a permutation Π over $[n]$ such that $\boldsymbol{v}'$ is the tuple resulting from applying Π to $\boldsymbol{v}$, i.e. $\mathsf{PerVec}_\Pi(\boldsymbol{v}) = \boldsymbol{v}'$, and vice versa. The equivalence class $[\boldsymbol{v}]_{\sim_n}$ is then defined as the set of all $\boldsymbol{v}' \in \mathbb{V}^n$ such that $\boldsymbol{v} \sim_n \boldsymbol{v}'$.

Definition 11 (Mix-Net-based Voting). *A voting scheme $\mathcal{V}ot$ is said to be a mix-net-based voting scheme if its result function F is defined as follows: given any n (distinct) eligible voters $\{\mathsf{id}_1, ..., \mathsf{id}_n\}$ and any tuple of votes $\boldsymbol{v} = (v_i)_{i\in[n]} \in \mathbb{V}^n$, $F((\mathsf{id}_i, v_i)_{i\in[n]})$ outputs the class $[\boldsymbol{v}]_{\sim_n}$.*

In the following, when the context is clear, we will only use $\sim$.

3.2 Corrective Fault Tolerance

We also assume that the votes of any eligible voters whose ballots are expected but missing from the ballot box, due to either non-submission or invalid verification, are treated as if they were not sent and will be represented by null values that can be identified and removed during the tallying process. In what follows, we refer to these voters as *dropout* voters. To handle these voters, we introduce the correcting keys $\{\mathsf{ck}_\mathsf{id}\}_\mathsf{id}$, which are generated by remaining voters. With the possible help of $\{\mathsf{ck}_\mathsf{id}\}_\mathsf{id}$, the final result can be publicly computed using the $\mathsf{SelfTally}$ algorithm. A formal definition is given below.

Definition 12 (Corrective Fault Tolerance). *A voting scheme $\mathcal{V}ot$ is said to adopt corrective fault tolerance if, for all parameters λ and all integers n, the following properties hold with overwhelming probability in λ: in the offline phase, given $(\mathsf{mpk}, \mathsf{msk})$ issued by $\mathsf{SetUp}(1^\lambda)$, $(\mathsf{pk}_i, \mathsf{ek}_i)$ issued by $\mathsf{KeyGen}(\mathsf{id}_i, \mathsf{msk}, \mathsf{mpk})$ for $i \in [n]$, $\mathsf{L} := (\mathsf{pk}_1, ..., \mathsf{pk}_n)$, and hpk issued by $\mathsf{PKProcess}(\mathsf{L}, \mathsf{msk})$, and in the online phase, for any strict subset $\mathsf{V} \subsetneq [n]$, let $(v_i)_{i\in\mathsf{V}} \in \mathbb{V}^{|\mathsf{V}|}$ be votes from voters $(\mathsf{id}_i)_{i\in\mathsf{V}}$, let b_i be ballots issued by $\mathsf{Vote}(\mathsf{ek}_i, v_i, \mathsf{hpk})$, let $\mathsf{BB} := \{b_i\}_{i\in\mathsf{V}}$, and let ck_i be correcting keys issued by $\mathsf{FaultCrt}(\mathsf{FaultAgg}(\mathsf{BB}, \mathsf{hpk}, \mathsf{id}_i), \mathsf{ek}_i)$ for all $i \in \mathsf{V}$. Then the tally is correct and valid: given $\mathsf{res} = \mathsf{SelfTally}(\mathsf{BB}, \{\mathsf{ck}_i\}_{i\in\mathsf{V}}, \mathsf{hpk})$, it holds that $\mathsf{res} = F((\mathsf{id}_i, v_i)_{i\in\mathsf{V}})$ and $\mathsf{Verify}(\mathsf{BB}, \{\mathsf{ck}_i\}_{i\in\mathsf{V}}, \mathsf{res}, \mathsf{L}, \mathsf{hpk}) = 1$.*

The above definition is adapted from the intuition in [KY02], where corrective actions are taken by honest voters. It is a correctness guarantee conditioned on every voter in V supplying a correctly generated correcting key, not a guarantee of liveness against malicious abstention during correction. In our constructions, missing required correcting keys prevent successful tallying, while invalid correcting-key proofs are rejected in the verifiable protocol. A recursive recovery procedure that changes the participating set would require a separate definition and analysis.

3.3 Client-Side Lightweightness

In large-scale settings, it is realistic to assume that voters use casual devices. Consequently, both computational and communication costs on the per-voter side should remain independent of the total number of voters. To capture this requirement, we introduce the notion of *client-side lightweightness* as a fundamental criterion for large-scale voting.

Definition 13 (Client-Side Lightweightness). *A voting scheme $\mathcal{V}ot$ over a universe $\mathcal{U}$ of $n_{\mathcal{U}}$ voters and a security parameter λ is said to be* client-side lightweight *if all algorithms involving the generation and use of a voter's private encryption key* $\mathsf{ek_{id}}$*, namely* $\mathsf{KeyGen}(\mathsf{id}, \mathsf{msk})$*,* $\mathsf{Vote}(\mathsf{ek_{id}}, v, \mathsf{hpk})$*,* $\mathsf{FaultCrt}(\mathsf{fa_{id}}, \mathsf{ek_{id}})$*, execute in time $O_\lambda(1)$ and incur a communication cost of $O_\lambda(1)$, where $O_\lambda(\cdot)$ hides polynomials in λ.*

To achieve client-side lightweightness in fault correction, it is necessary to assume that the ballot box manager behaves as an honest-but-curious authority. This assumption is justified for two reasons. First, the operations executed by the Append and $\mathsf{FaultAgg}$ algorithms are transparent, deterministic and hence publicly verifiable via a log. Second, in the absence of an honest ballot box manager, honest voters will have to verify the voters declared to have dropped out on the bulletin board to prevent false accusations. This setting immediately precludes the possibility of client-side lightweightness.

3.4 Ballot Privacy

In this work, our definition of ballot privacy is inspired by the one given by Benaloh in [Ben87]. Intuitively, a voting scheme is private if no adversary is able to distinguish between two sets of challenge votes that are tallied into the same result, given the election parameter, the ballot box, and the selection of voters to be corrupted. As in the voting syntax, we define ballot privacy for single-vote systems, where each voter is expected to submit at most one ballot.

Definition 14 (Ballot Privacy). *Let $\mathcal{V}ot$ be a voting scheme for a universe of voters $\mathcal{U}$, a vote space $\mathbb{V}$ and a result function $F : (\mathcal{U} \times \mathbb{V})^* \to \{0,1\}^*$. We define the privacy experiment $\mathsf{Exp}_\beta^{\mathsf{SelfPriv}}$ in Figure 6, where $\beta \in \{0,1\}$ and the oracles are defined as follows:*

- $\mathsf{QNewHonest}(\mathsf{id})$*: If* id *is already registered, i.e.* $\mathsf{id} \in \mathsf{L}$*, the query is ignored. Otherwise, on input an identity* $\mathsf{id} \in \mathcal{U}$*, the challenger generates* $(\mathsf{pk_{id}}, \mathsf{ek_{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{id}, \mathsf{msk})$ *and runs* $\mathsf{Register}(\mathsf{L}, \mathsf{pk_{id}})$*. Once registration succeeds, it stores the association* $(\mathsf{ek_{id}}, \mathsf{pk_{id}})$*.*
- $\mathsf{QCor}(\mathsf{id})$*: The challenger recovers the private encryption key* $\mathsf{ek_{id}}$ *associated to* $\mathsf{pk_{id}}$ *and returns it to the adversary. If no such association is found, the query is ignored.*
- $\mathsf{QReqCast}(\mathsf{id}, \mathsf{pk_{id}})$*: If* $\mathsf{id} \notin \mathsf{L}$*, on input a public key* $\mathsf{pk_{id}}$*, the challenger runs* $\mathsf{Register}(\mathsf{L}, \mathsf{pk_{id}})$*. Otherwise, the query is ignored.*
- $\mathsf{QPKProcess}()$*: This oracle is used only once in the game; the challenger outputs* $\mathsf{hpk} \leftarrow \mathsf{PKProcess}(\mathsf{L}, \mathsf{msk})$ *to the adversary. If* $\mathsf{PKProcess}(\mathsf{L}, \mathsf{msk})$ *is a protocol between the adversary (as corrupted trustees) and the challenger (as honest trustees), then a query asks the challenger to use* msk *to join this protocol. Once a query is made, no* $\mathsf{QNewHonest}$ *or* $\mathsf{QReqCast}$ *queries are allowed.*

The following oracles can be queried after a query is sent to $\mathsf{QPKProcess}()$*.*

- $\mathsf{QVote}_\beta(\mathsf{id}, v^0, v^1)$*: For a voter* id*, if $v^0, v^1 \in \mathbb{V}$ and the private encryption key* $\mathsf{ek_{id}}$ *can be recovered, the challenger generates* $b_{\mathsf{id}} \leftarrow \mathsf{Vote}(\mathsf{ek_{id}}, v^\beta, \mathsf{hpk})$ *and runs* $\mathsf{Append}(\mathsf{BB}, b_{\mathsf{id}}, \mathsf{L}, \mathsf{hpk})$*. Once appending to the ballot box succeeds, the query is recorded. Otherwise, the query is ignored.*
- $\mathsf{QVotCast}(\mathsf{id}, b_{\mathsf{id}})$*: If* id *is corrupted via* QCor *or registered via* $\mathsf{QReqCast}$[9]*, this oracle allows the adversary to cast a ballot* b_{id} *on behalf of voter* id*; the challenger then runs* $\mathsf{Append}(\mathsf{BB}, b_{\mathsf{id}}, \mathsf{L}, \mathsf{hpk})$*. Otherwise, the query is ignored.*
- $\mathsf{QFaultCrt}()$*: This oracle can be queried only once in the game. The challenger returns the correcting keys* $\mathsf{ck_{id}} \leftarrow \mathsf{FaultCrt}(\mathsf{FaultAgg}(\mathsf{BB}, \mathsf{hpk}, \mathsf{id}), \mathsf{ek_{id}})$ *for all* id *such that* $(\mathsf{ek_{id}}, \mathsf{pk_{id}})$ *is stored. No further oracle queries are allowed.*

$\mathsf{Exp}_\beta^{\mathsf{SelfPriv}}(\lambda, \mathcal{U}, F, \mathcal{A})$:
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\lambda)$, $\mathsf{BB}, \mathsf{L} \leftarrow \emptyset$;
$\beta' \leftarrow \mathcal{A}^{\mathsf{QNewHonest}(\cdot),\mathsf{QCor}(\cdot),\mathsf{QReqCast}(\cdot,\cdot),\mathsf{QPKProcess}(),\mathsf{QVote}_\beta(\cdot,\cdot,\cdot),\mathsf{QVotCast}(\cdot,\cdot),\mathsf{QFaultCrt}()}(\mathsf{mpk}, \mathsf{BB}, \mathsf{L})$;
output $\alpha = \beta'$ if condition $(*)$ is satisfied and $\alpha \xleftarrow{\$} \{0,1\}$ otherwise.

Fig. 6: Experiment for ballot privacy

[9] In Definition 9, we assume the public correspondence between an identity id and its ballot b_{id}, therefore the cases an adversary casts ballots on behalf of honest voters can be ruled out.

Here, β is the challenge bit, β' is the adversary's output bit, and α is the experiment's output bit; the experiment does not test whether $\beta' = \beta$.

At the time the QFaultCrt *query is made, let* L *be the list of eligible voters and let $\mathcal{HS}$ be the subset of honest voters (whose public keys are generated via* QNewHonest *but not corrupted via* QCor*) with recorded* QVote_β *queries.*

Condition $(*)$, which prevents trivial wins, holds if: there does not exist any tuple of votes $(\mathsf{id}, v_{\mathsf{id}})_{\mathsf{id}\in\mathsf{L}\setminus\mathcal{HS}}$ such that

$$F((\mathsf{id}, v^0_{\mathsf{id}})_{\mathsf{id}\in\mathcal{HS}}, (\mathsf{id}, v_{\mathsf{id}})_{\mathsf{id}\in\mathsf{L}\setminus\mathcal{HS}}) \neq F((\mathsf{id}, v^1_{\mathsf{id}})_{\mathsf{id}\in\mathcal{HS}}, (\mathsf{id}, v_{\mathsf{id}})_{\mathsf{id}\in\mathsf{L}\setminus\mathcal{HS}}),$$

where for each $\mathsf{id} \in \mathcal{HS}$, the query $\mathsf{QVote}(\mathsf{id}, v^0_{\mathsf{id}}, v^1_{\mathsf{id}})$ was recorded[10]*.*

The scheme $\mathcal{V}ot$ is said to have ballot privacy with respect to the result function F if for any PPT adversary $\mathcal{A}$, the following advantage is negligible in λ:

$$\mathsf{Adv}^{\mathsf{SelfPriv}}(\lambda, \mathcal{A}) := \left|\Pr[\mathsf{Exp}^{\mathsf{SelfPriv}}_0(\lambda, \mathcal{U}, F, \mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{SelfPriv}}_1(\lambda, \mathcal{U}, F, \mathcal{A}) = 1]\right|$$

Remark 1 (Restricted Variants). The scheme $\mathcal{V}ot$ might have ballot privacy in a weaker setting if the following additional conditions are added to Condition $(*)$:

- *Symmetric-key* privacy: Let $\mathcal{CS}$ be the set of voters that are corrupted via QCor at the end of the game. If $\mathsf{id} \in \mathcal{CS}$, then for any queries $\mathsf{QVote}(\mathsf{id}, v^0, v^1)$, it must hold that $v^0 = v^1$.
- *Selective* privacy: Once receiving mpk, the adversary is forced to send all QNewHonest queries, upon which it receives the corresponding public keys of honest voters. Then it commits all QCor and QVote_β queries in one shot.

In comparison to Benaloh's definition, we modify the restriction on adversaries so that the result function, when evaluated on challenge votes of honest users and *any possible* non-challenge votes, must produce an output independent of the challenge bit. This can be verified for common functions such as homomorphic tallying and mix-net based tallying. Furthermore, under this modified restriction, a voting scheme for the majority function might be considered ballot-private, unlike in Benaloh's definition, as noted in [BCG+15]. On the other hand, the simulated tallying proofs that are enabled in the definition in [BCG+15], are non-applicable in settings where tallying can be executed publicly.

3.5 Verifiability

In a voting system, verifiability ensures that a voter can verify, even in the presence of dishonest authorities, that their ballot has been correctly included in the final tally (individual verifiability), that the result reflects the intended votes of voters as published in the ballot box (universal verifiability), and that only ballots from eligible voters are counted (eligibility verifiability).

In this work, eligibility verifiability is ensured through implicit credentials issued by the registrar and a publicly available list of eligible voters. Individual verifiability is provided by the public nature of the ballot box, allowing each voter to confirm the presence of their own ballot. We provide a formalisation of universal verifiability in Definition 15, which captures the integrity of honest voters' votes in the final results.

Definition 15 (Universal Verifiability). *A voting scheme $\mathcal{V}ot$ for a universe of voters $\mathcal{U}$ and a result function $F : (\mathcal{U} \times \mathbb{V})^* \to \{0,1\}^*$ satisfies universal verifiability if, for any PPT adversary $\mathcal{A}$, the following experiment is sound with overwhelming probability:*

- *$\mathcal{A}$ outputs a master public key* mpk*, and selects the honest-voter set $\mathcal{HS}$.*
- *The challenger computes $(\mathsf{pk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{id}, \mathsf{msk}, \mathsf{mpk})$ (possibly with $\mathcal{A}$ when a master secret key* msk *is needed as input) for $\mathsf{id} \in \mathcal{HS}$, posts $\{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{HS}}$ to a list of eligible voters* L *and stores the private encryption keys $\{\mathsf{ek}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{HS}}$.*
- *On input $\{\mathsf{pk}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{HS}}$, $\mathcal{A}$ completes the list* L *and outputs a helper public key* hpk*.*
- *On input valid votes $v_{\mathsf{id}} \in \mathbb{V}$ for $\mathsf{id} \in \mathcal{HS}$ chosen by $\mathcal{A}$, the challenger generates ballots $b_{\mathsf{id}} \leftarrow \mathsf{Vote}(\mathsf{ek}_{\mathsf{id}}, v_{\mathsf{id}}, \mathsf{hpk})$ and posts them to a ballot box* BB*.*
- *On input $(b_{\mathsf{id}})_{\mathsf{id}\in\mathcal{HS}}$, $\mathcal{A}$ completes the box* BB*.*
- *On input a complete box* BB*, the challenger generates correcting keys $\mathsf{ck}_{\mathsf{id}} \leftarrow \mathsf{FaultCrt}(\mathsf{FaultAgg}(\mathsf{BB}, \mathsf{hpk}, \mathsf{id}), \mathsf{ek}_{\mathsf{id}})$ for $\mathsf{id} \in \mathcal{HS}$.*
- *On input $\{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{HS}}$, $\mathcal{A}$ completes the list of correcting keys $\{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}$.*

[10] The adversary can access only a unique ballot for each voter in $\mathcal{HS}$ via BB, due to the no-revote policy.

– $\mathcal{A}$ *finalises the game with a result* res.

Given the ballot box BB *containing* $\{b_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{HS}}$ *and* $\{b_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{CS}}$ *for some malicious-voter set* $\mathcal{CS}$*, if*

$$\mathsf{Verify}(\mathsf{BB}, \{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{HS}\cup\mathcal{CS}}, \mathsf{res}, \mathsf{L}, \mathsf{hpk}) = 1,$$

then there exists a subset $\mathcal{CS}' \subset \mathcal{CS}$ *and* $v_{\mathsf{id}} \in \mathbb{V}$ *for* $\mathsf{id} \in \mathcal{CS}'$ *such that*

$$\mathsf{res} = F((\mathsf{id}, v_{\mathsf{id}})_{\mathsf{id}\in\mathcal{HS}}, (\mathsf{id}, v_{\mathsf{id}})_{\mathsf{id}\in\mathcal{CS}'}).$$

This notion is strong in the sense that if the verification succeeds, intended votes from honest voters are always included in the final result, along with only valid votes from malicious voters.

We complete the voting definitions with the informal notions of fairness and dispute-freeness in Section 3.6.

3.6 Other Notions

We discuss two further properties from the self-tallying literature [KY02, Gro04a, HRZ10, MSH17]. They are not additional security theorems for our construction.

– **Fairness**: Informally, no party should learn a partial tally before the voting deadline. Self-tallying creates a last-voter problem: the final voter may learn the tally before publishing its own ballot. As discussed in [MSH17], the approach of [KY02, Gro04a] uses an authority to cast a final vote excluded from the tally, introducing trust in its cooperation and non-collusion. An alternative commitment phase fixes votes before they are revealed, but does not itself prevent the last voter from learning the tally early [MSH17]. Our protocol does not include a separately analysed fairness mechanism, so we do not claim deadline-based fairness. Adopting a delayed dummy-ballot mechanism would require specifying its generation, exclusion from the result, release policy, and interaction with dropout correction. Proofs of ballot validity alone do not enforce delayed release or non-collusion.
– **Dispute-freeness**: In the traditional informal sense, anyone can check whether voters follow the protocol; this includes public verification of the result [HRZ10]. For our construction, the formal claim is universal verifiability as defined in Definition 15, supported by public checks of ballots and correcting keys. We do not infer from this a general dispute-resolution or blame mechanism, nor a guarantee that missing messages will be supplied. In particular, publicly detecting a missing correcting key does not ensure completion of the tally.

Thus, self-tallying and universal verifiability should not be read as guarantees of fairness or liveness against withholding. The voting protocol in Figures 9 and 10 is analysed under the participation and trust assumptions stated above.

4 Voting with Functional Encryption

4.1 Preliminaries on Functional Encryption

We recall functional encryption, introduced in [SW05, O'N10, BSW11], in the function-hiding inner-product setting of [BJK15, TAO16].

Definition 16 (Pairing-based FH-IPFE). *Given a pairing group* $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p)$*, a functional encryption for hiding inner-product functions over* $\mathcal{PG}$ *consists of four algorithms* $\mathsf{IPE} = (\mathsf{SetUp}, \mathsf{DKGen}, \mathsf{Enc}, \mathsf{Dec})$*:*

– $\mathsf{SetUp}(1^\lambda, 1^\rho)$*: On input a security parameter* λ *and an inner product length* ρ*, it outputs public parameters* pp *and a secret key* sk*. The parameter* pp *is an implicit argument to other algorithms.*
– $\mathsf{DKGen}(\mathsf{sk}, [\![\boldsymbol{k}]\!]_1)$*: On input a secret key* sk *and a function that is represented by a key vector* $[\![\boldsymbol{k}]\!]_1 \in \mathbb{G}_1^\rho$*, it outputs a decryption key* $\mathsf{dk}_{\boldsymbol{k}}$*.*
– $\mathsf{Enc}(\mathsf{sk}, [\![\boldsymbol{m}]\!]_2)$*: On input a secret key* sk *and a message vector* $[\![\boldsymbol{m}]\!]_2 \in \mathbb{G}_2^\rho$*, it outputs a ciphertext* $\mathsf{ct}_{\boldsymbol{m}}$*.*
– $\mathsf{Dec}(\mathsf{dk}, \mathsf{ct})$*: On input a decryption key* dk *and a ciphertext* ct*, it outputs an element in* $\mathbb{G}_T$ *or an error message* $\perp$*.*

Correctness For all parameters $\lambda, \rho \in \mathbb{N}$, all secret keys sk generated by $\mathsf{SetUp}(1^\lambda, 1^\rho)$, all key vectors $\boldsymbol{k} \in \mathbb{Z}_p^\rho$ and all message vectors $\boldsymbol{m} \in \mathbb{Z}_p^\rho$, given $\mathsf{dk} = \mathsf{DKGen}(\mathsf{sk}, [\![\boldsymbol{k}]\!]_1)$ and $\mathsf{ct} = \mathsf{Enc}(\mathsf{sk}, [\![\boldsymbol{m}]\!]_2)$, it holds that $\mathsf{Dec}(\mathsf{dk}, \mathsf{ct}) = [\![\langle \boldsymbol{k}, \boldsymbol{m} \rangle]\!]_T$.

At a high level, the security ensures that given any $\mathsf{dk}_{\boldsymbol{k}}$ and $\mathsf{ct}_{\boldsymbol{m}}$, no information beyond $[\![\langle \boldsymbol{k}, \boldsymbol{m} \rangle]\!]_T$ can be learned; a game-based definition is provided in Definition 17.

Definition 17 (Function-Hiding Privacy). *For a functional encryption scheme* IPE *for hiding inner products of length* ρ *over a pairing group* $\mathcal{PG}$*, we define the experiment* $\mathrm{IND}_b^{\mathsf{IPE}}$ *in Figure 7, where* $b \in \{0, 1\}$ *and the oracles are defined as:*

- ***Decryption-key oracle*** $\mathsf{QDKGen}_b([\![\boldsymbol{k}^0]\!]_1, [\![\boldsymbol{k}^1]\!]_1)$*: it returns the decryption key* $\mathsf{dk} \leftarrow \mathsf{DKGen}(\mathsf{sk}, [\![\boldsymbol{k}^b]\!]_1)$.
- ***Encryption oracle*** $\mathsf{QEnc}_b([\![\boldsymbol{m}^0]\!]_2, [\![\boldsymbol{m}^1]\!]_2)$*: it returns the ciphertext* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, [\![\boldsymbol{m}^b]\!]_2)$.
- *Condition* $(*)$*: Every decryption-key query on* $([\![\boldsymbol{k}^0]\!]_1, [\![\boldsymbol{k}^1]\!]_1)$ *and every encryption query on* $([\![\boldsymbol{m}^0]\!]_2, [\![\boldsymbol{m}^1]\!]_2)$ *satisfy* $[\![\langle \boldsymbol{k}^0, \boldsymbol{m}^0 \rangle]\!]_T = [\![\langle \boldsymbol{k}^1, \boldsymbol{m}^1 \rangle]\!]_T$.

$\underline{\mathrm{IND}_b^{\mathsf{IPE}}(\lambda, \rho, \mathcal{A})}$:
$(\mathsf{pp}, \mathsf{sk}) \leftarrow \mathsf{SetUp}(1^\lambda, 1^\rho); \qquad \alpha \leftarrow \mathcal{A}^{\mathsf{QEnc}_b(\cdot,\cdot), \mathsf{QDKGen}_b(\cdot,\cdot)}(\mathsf{pp})$
Output: $\beta = \alpha$ if Condition $(*)$ is satisfied, $\beta \xleftarrow{\$} \{0, 1\}$ otherwise.

Fig. 7: Security experiment for IPE

The IPE *scheme is said to be IND-CPA secure if given any parameter* $\rho \in \mathbb{N}$*, for every PPT adversary* $\mathcal{A}$*, the following advantage is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{IND}^{\mathsf{IPE}}}(\lambda, \rho, \mathcal{A}) := \Big|\Pr[\mathrm{IND}_0^{\mathsf{IPE}}(\lambda, \rho, \mathcal{A}) = 1] - \Pr[\mathrm{IND}_1^{\mathsf{IPE}}(\lambda, \rho, \mathcal{A}) = 1]\Big|.$$

Definition 18 (Homomorphic FE Key). *An* IPE *scheme supports homomorphic functional keys if, given any two keys* $\mathsf{dk}_1 = \mathsf{DKGen}(\mathsf{sk}, [\![\boldsymbol{k}_1]\!]_1; \boldsymbol{r}_1)$ *and* $\mathsf{dk}_2 = \mathsf{DKGen}(\mathsf{sk}, [\![\boldsymbol{k}_2]\!]_1; \boldsymbol{r}_2)$ *(in the form of vectors in* $\mathbb{G}$*) under randomness* $\boldsymbol{r}_1$ *and* $\boldsymbol{r}_2$ *(in the form of vectors in* $\mathbb{Z}_p$*) respectively, for any scalars* $\alpha_1, \alpha_2 \in \mathbb{Z}_p$*, the following holds:*

$$\alpha_1 \cdot \mathsf{dk}_1 + \alpha_2 \cdot \mathsf{dk}_2 = \mathsf{DKGen}(\mathsf{sk}, [\![\alpha_1 \boldsymbol{k}_1 + \alpha_2 \boldsymbol{k}_2]\!]_1; \alpha_1 \boldsymbol{r}_1 + \alpha_2 \boldsymbol{r}_2).$$

Schemes realizing the above definitions are given in [TAO16] and [Lin17], and are proven secure under the Decisional Linear (DLIN) assumption and the Symmetric External Diffie–Hellman (SXDH) assumption, respectively. Both assumptions can be viewed as particular cases of the standard parameterized k-Lin assumption in prime-order bilinear groups.

4.2 A Warm-up Protocol

By leveraging functional encryption for function-hiding inner products, we present a mix-net-based voting protocol that incorporates the generation of a permutation and its corresponding decryption key in the offline phase. For simplicity, our conceptual protocol assumes an authority that serves as both an election administrator and a trustee. We also assume secure, authenticated private channels between the authority and each voter for generating private encryption keys.

In this conceptual protocol, we do not aim yet for universal verifiability. This simplification allows us to focus on the role of functional encryption in ensuring ballot privacy while enabling a decryption key for shuffling to be generated in the offline phase, thereby removing the need for any trusted authority in the tallying phase. Furthermore, the ballot privacy proof for this protocol serves as a foundation for our extended construction, which incorporates universal verifiability and an offline mix-net with multiple servers.

Assuming an FH-IPFE scheme with $(\mathsf{ip.Setup}, \mathsf{ip.DKGen}, \mathsf{ip.Enc}, \mathsf{ip.Dec})$ algorithms in an asymmetric pairing group $\mathcal{PG}$ of order p, we present our warm-up protocol for the mixing result function in Figure 8. The vote space is defined as $\mathbb{V} = \{1, \ldots, V\} \subset \mathbb{Z}_p$ for some λ-polynomially large integer V. In this version, we omit the Verify algorithm.

We now show correctness, corrective fault tolerance and client-side lightweightness.

$\underline{\mathsf{SetUp}(1^\lambda)}$: The authority generates random $t_{\mathsf{id}} \xleftarrow{\$} \mathbb{Z}_p$ and FH-IPFE secret keys $\mathsf{ip.sk}_{\mathsf{id}} \leftarrow \mathsf{ip.Setup}(1^\lambda, 1^3)$, and stores $\mathsf{ek}_{\mathsf{id}} = (\mathsf{ip.sk}_{\mathsf{id}}, t_{\mathsf{id}})$ for $\mathsf{id} \in \mathcal{U}$ in msk. It outputs mpk as the public parameter for FH-IPFE (including a pairing group $\mathcal{PG}$) and the vote space $\mathbb{V} = \{1, \ldots, V\}$.
$\underline{\mathsf{KeyGen}(\mathsf{id}, \mathsf{msk}, \mathsf{mpk})}$: Via a private and authenticated channel, voter id receives $\mathsf{ek}_{\mathsf{id}} = (\mathsf{ip.sk}_{\mathsf{id}}, t_{\mathsf{id}})$ from the authority; $\mathsf{pk}_{\mathsf{id}}$ contains $[\![t_{\mathsf{id}}]\!]_2$.

$\underline{\mathsf{Register}(\mathsf{L}, \mathsf{pk}_{\mathsf{id}})}$: The authority adds $\mathsf{pk}_{\mathsf{id}}$ to the list of eligible voters L.

$\underline{\mathsf{PKProcess}(\mathsf{L}, \mathsf{msk})}$: For n voters in $\mathsf{L} = \{\mathsf{id}_i\}_{i\in[n]}$, the authority generates a random permutation $\Pi : [n] \to [n]$, a one-time-pad vector $\boldsymbol{z} = (z_j)_{j\in[n]} \xleftarrow{\$} \mathbb{Z}_p^n$, and for each $i \in [n]$:

1. an aggregate public key for zero sharing: $\mathsf{pz}_i = \sum_{k\in[n]\setminus\{i\}} (-1)^{k<i} \cdot [\![t_k]\!]_2$
2. a functional key for each $j \in [n]$: $\mathsf{ip.dk}_j^i \leftarrow \mathsf{ip.DKGen}(\mathsf{ip.sk}_i, [\![p_j^i, z_j, 0]\!]_1)$ where $\boldsymbol{p}^i$ is a vector that represents $\Pi(i)$ such that $p_j^i = 1$ if $j = \Pi(i)$ and $p_j^i = 0$ otherwise;
3. components for later extension purposes $\mathsf{aux}_i = (\mathsf{ip.dk}_{(0,1)}^i, \mathsf{ip.dk}_{(0,0)}^i)$ where

$$\mathsf{ip.dk}_{(0,1)}^i = \mathsf{ip.DKGen}(\mathsf{ip.sk}_i, [\![0, 1, 0]\!]_1)$$
$$\mathsf{ip.dk}_{(0,0)}^i = \mathsf{ip.DKGen}(\mathsf{ip.sk}_i, [\![0, 0, 0]\!]_1).$$

It publishes $\mathsf{hpk} = \mathsf{mpk} || \{\mathsf{id}_i, \mathsf{pk}_i, (\mathsf{ip.dk}_j^i)_{j\in[n]}, \mathsf{pz}_i, \mathsf{aux}_i\}_{i\in[n]} || [\![\boldsymbol{z}]\!]_1$.

$\underline{\mathsf{Vote}(\mathsf{ek}_{\mathsf{id}}, v_{\mathsf{id}}, \mathsf{hpk})}$: For $v_{\mathsf{id}} \in \mathbb{V}$, voter id computes an FH-IPFE encryption:

$$\mathsf{ip.ct}^{\mathsf{id}} \leftarrow \mathsf{ip.Enc}(\mathsf{ip.sk}_{\mathsf{id}}, ([\![v_{\mathsf{id}}]\!]_2, t_{\mathsf{id}} \cdot \mathsf{pz}_{\mathsf{id}}, [\![0]\!]_2));$$

and casts $b_{\mathsf{id}} = \mathsf{ip.ct}^{\mathsf{id}}$ to BB. Registered voter id can abort voting.

$\underline{\mathsf{Append}(\mathsf{BB}, b_{\mathsf{id}}, \mathsf{L}, \mathsf{hpk})}$: If $\mathsf{id} \in \mathsf{L}$ and there does not exist $b'_{\mathsf{id}} \in \mathsf{BB}$, the authority appends b_{id} to BB.

$\underline{\mathsf{FaultAgg}(\mathsf{BB}, \mathsf{hpk}, \mathsf{id})}$: Let $\mathsf{V} \subseteq [n]$ be the list of each $\mathsf{id}_i \in \mathsf{L}$ whose ballot is contained in BB. If $\mathsf{V} = [n]$, nothing is output. Otherwise, the fault aggregate for the voter $i \in \mathsf{V}$ that corresponds to id is $\mathsf{fa}_i = \sum_{k\in[n]\setminus\mathsf{V}} (-1)^{k<i} \cdot [\![t_k]\!]_2$.

$\underline{\mathsf{FaultCrt}(\mathsf{fa}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}})}$: The correcting key is output as $\mathsf{ck}_{\mathsf{id}} = t_{\mathsf{id}} \cdot \mathsf{fa}_{\mathsf{id}}$.

$\underline{\mathsf{SelfTally}(\mathsf{BB}, \{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}, \mathsf{hpk})}$: Let $\mathsf{V} \subseteq [n]$ be the list of voters $\mathsf{id}_i \in \mathsf{L}$ whose ballots are contained in BB and for whom ck_i is given (in case $\mathsf{V} \subsetneq [n]$)[a], and let $\boldsymbol{v}^\star$ be an empty tuple. It obtains $\{\mathsf{ip.dk}_j^i\}_{j,i\in[n]}$ from hpk, $\{\mathsf{ip.ct}^i\}_{i\in\mathsf{V}}$ from BB. For each $j \in [n]$, it recovers v'_j from $[\![v'_j]\!]_T = \sum_{i\in\mathsf{V}} \mathsf{ip.Dec}(\mathsf{ip.dk}_j^i, \mathsf{ip.ct}^i) - e([\![z_j]\!]_1, \sum_{i\in\mathsf{V}} \mathsf{ck}_i)$ and appends v'_j to $\boldsymbol{v}^\star$ if $[\![v'_j]\!]_T \neq [\![0]\!]_T$[b]. The result res is the equivalence class $[\boldsymbol{v}^\star]_{\sim_{|\mathsf{V}|}}$.

[a] If such a list does not exist, the algorithm outputs $\perp$ as the result.
[b] $v'_j = 0$ corresponds to a null vote $\perp$ in Definition 9.

Fig. 8: A conceptual voting protocol with FH-IPFE

Correctness. Using the notation from the voting protocol, one can parse columns $(\boldsymbol{p}^i)_{i\in[n]}$ as the identity matrix with rows permuted by Π, or $\mathsf{PerMatRow}_\Pi(\boldsymbol{I})$. Assuming all eligible voters send their ballots, we have

$$\begin{aligned}
[\![\boldsymbol{v}']\!]_T &= (\sum_{i\in[n]} \mathsf{ip.Dec}(\mathsf{idk}_j^i, \mathsf{ict}_i))_{j\in[n]} = (\sum_{i\in[n]} [\![v_i p_j^i + z_j \sum_{k\in[n]\setminus\{i\}} (-1)^{k<i} t_k t_i]\!]_T)_{j\in[n]} \\
&= ([\![\sum_{i\in[n]} v_i p_j^i + \sum_{i,k\in[n],i\neq k} \left((-1)^{k<i} + (-1)^{i<k}\right) t_k t_i]\!]_T)_{j\in[n]} \\
&= ([\![\sum_{i\in[n]} v_i p_j^i]\!]_T)_{j\in[n]} = [\![\mathsf{PerMatRow}_\Pi(\boldsymbol{I})\boldsymbol{v}]\!]_T = [\![\mathsf{PerVec}_\Pi(\boldsymbol{v})]\!]_T
\end{aligned}$$

As $\boldsymbol{v} \in \mathbb{V}^n$ and $[\![\boldsymbol{v}']\!]_T$ can be recovered by the computation of discrete logarithms, we have $\boldsymbol{v}^\star = \mathsf{PerVec}_\Pi(\boldsymbol{v})$. Therefore, $[\boldsymbol{v}^\star]_{\sim_n} = [\boldsymbol{v}]_{\sim_n}$.

Corrective Fault Tolerance. One can parse columns $(\boldsymbol{p}^i)_{i\in[n]}$ as the identity matrix with rows permuted by Π, or $\mathsf{PerMatRow}_\Pi(\boldsymbol{I})$. Assuming voters in $\mathsf{V} \subset [n]$ send their ballots and correcting keys, we have

$$\begin{aligned}
[\![\boldsymbol{v}']\!]_T &= (\sum_{i\in\mathsf{V}} \mathsf{ip.Dec}(\mathsf{idk}_j^i, \mathsf{ict}_i) - \sum_{i\in\mathsf{V}} e([\![z_j]\!]_1, \mathsf{ck}_i))_{j\in[n]} \\
&= (\sum_{i\in\mathsf{V}} [\![v_i p_j^i + z_j \sum_{k\in[n]\setminus\{i\}} (-1)^{k<i} t_k t_i - z_j \sum_{k\in[n]\setminus\mathsf{V}} (-1)^{k<i} t_k t_i]\!]_T)_{j\in[n]} \\
&= (\sum_{i\in\mathsf{V}} [\![v_i p_j^i + z_j \sum_{k\in\mathsf{V}\setminus\{i\}} (-1)^{k<i} t_k t_i]\!]_T)_{j\in[n]} \\
&= ([\![\sum_{i\in\mathsf{V}} v_i p_j^i + \sum_{i,k\in\mathsf{V},i\neq k} \left((-1)^{k<i} + (-1)^{i<k}\right) t_k t_i]\!]_T)_{j\in[n]} \\
&= ([\![\sum_{i\in\mathsf{V}} v_i p_j^i]\!]_T)_{j\in[n]}
\end{aligned}$$

Then we have $v'_j = v_i$ if $j = \Pi(i)$ for $i \in \mathsf{V}$, and $v'_j = 0$ if $j = \Pi(i)$ for $i \notin \mathsf{V}$. When zero entries of $\boldsymbol{v}'$ are removed, we have the $\mathsf{SelfTally}$ output $[\boldsymbol{v}^\star]_{\sim_{|\mathsf{V}|}} = [\boldsymbol{v}']_{\sim_{|\mathsf{V}|}}$.

Client-Side Lightweightness. The protocol guarantees that both the per-voter communication and computation costs remain constant with respect to the total number of voters. Specifically, since the number of operations in each FH-IPFE algorithm is $O_\lambda(1)$, i.e. independent of the number of voters, we have: in $\mathsf{KeyGen}(\mathsf{id}, \mathsf{msk}, \mathsf{mpk})$, the private encryption key $\mathsf{ek}_\mathsf{id} = (\mathsf{ip.sk}_\mathsf{id}, t_\mathsf{id})$ and the public key $\mathsf{pk}_\mathsf{id} = [\![t_\mathsf{id}]\!]_2$ are of size $O_\lambda(1)$; in $\mathsf{Vote}(\mathsf{ek}_\mathsf{id}, v_\mathsf{id}, \mathsf{hpk})$, the ballot b_id has size $O_\lambda(1)$, and the computation cost is that of a single FH-IPFE encryption; in $\mathsf{FaultCrt}(\mathsf{fa}_\mathsf{id}, \mathsf{ek}_\mathsf{id})$, the correcting key ck_id is a single group element in $\mathbb{G}_2$ and is computed with one exponentiation. Hence, the protocol is client-side lightweight, as defined in Definition 13.

4.3 Ballot Privacy

For the security analysis of the construction, we need the following remark.

Remark 2. Condition $(*)$ in ballot privacy (in Definition 14) when applied to mix-net-based voting (Definition 11) implies that the equivalence class represented by $(v^0_\mathsf{id})_{\mathsf{id}\in\mathcal{HS}}$ is equal to that represented by $(v^1_\mathsf{id})_{\mathsf{id}\in\mathcal{HS}}$, where $(\mathsf{id}, v^0_\mathsf{id}, v^1_\mathsf{id})_{\mathsf{id}\in\mathcal{HS}}$ are the recorded vote queries on a subset $\mathcal{HS}$ of honest voters. In other words, there must exist a bijection Π over $\mathcal{HS}$ such that $v^0_{\Pi(\mathsf{id})} = v^1_\mathsf{id}$.

Theorem 1. *If FH-IPFE is IND-CPA secure and the SXDH assumption holds in $\mathcal{PG}$, then the warm-up scheme for mix-net voting is symmetric-key and selectively secure, as per Definition 14.*

Proof. At the end of the game, let L be the list of eligible voters, $\mathcal{CS}$ be the set of voters that are corrupted via QCor, and $\mathcal{HS}$ be the subset of honest voters (whose public keys are generated via $\mathsf{QNewHon}$ but not corrupted via QCor) that are queried and recorded in QVote_β. In the selective setting, $\mathcal{HS}$, $\mathcal{CS}$, and $\{(\mathsf{id}, v^0, v^1)\}_{\mathsf{id}\in\mathcal{HS}}$ are specified beforehand by the $\mathsf{QNewHon}$, QCor, and QVote_β queries. In the symmetric-key setting, any (id, v^0, v^1) vote queries for $\mathsf{id} \in \mathcal{CS}$ must satisfy $v^0 = v^1$.

To align with permutations, we assume $n = |\mathsf{L}|$ and index the voters in L using integers from $[n]$. From this point forward, we denote $\mathcal{HI}$ as the set of indices in $[n]$ corresponding to the voters in $\mathcal{HS}$. The index set $[n] \setminus \mathcal{HI}$ thus corresponds to the voters in $\mathsf{L} \setminus \mathcal{HS}$.

It suffices to prove that it is hard for $\mathcal{A}$ to distinguish between the real game where QVote_β queries are answered based on β and the ideal game where QVote_β queries are answered based on a fixed bit, for example 0. We proceed via a hybrid argument where $\mathbf{G}_0$ corresponds to the real security game and $\mathbf{G}_5$ corresponds to the ideal one. We highlight the changes between consecutive games in gray boxes.

Game $\mathbf{G}_1$: If the vote queries $(i, v_i^0, v_i^1)_{i\in\mathcal{HI}}$ satisfy Condition $(*)$ in the security game, then by Remark 2 there exists a bijection Π^β over $\mathcal{HI}$ such that $v_i^\beta = v^0_{\Pi^\beta(i)}$. This bijection can be extended to a permutation over $[n]$ by maintaining $\Pi^\beta(i) = i$ for $i \in [n] \setminus \mathcal{HI}$. The challenger generates a random $\Pi \circ \Pi^\beta$ instead of a random Π in the $\mathsf{QPKProcess}$ oracle. The indistinguishability between $\mathbf{G}_0$ and $\mathbf{G}_1$ is thus perfect.

We have the following remark on the relation between two permutations $(\Pi \circ \Pi^\beta, \Pi)$ and the vote queries $(i, v_i^0, v_i^1)_{i\in\mathcal{HI}}$

Remark 3. As $\Pi \circ \Pi^\beta(i) = \Pi(i)$ for $i \in [n] \setminus \mathcal{HI}$, these two permutations then have their representing column vectors $(\boldsymbol{p}^{\beta,i})_{i\in[n]}$ and $(\boldsymbol{p}^{0,i})_{i\in[n]}$ such that $\boldsymbol{p}^{\beta,i} = \boldsymbol{p}^{0,i}$ for $i \in [n] \setminus \mathcal{HI}$. Given any $(v_i^\beta)_{i\in[n]\setminus\mathcal{HI}} = (v_i^0)_{i\in[n]\setminus\mathcal{HI}}$, then permuting vector $\boldsymbol{v}^\beta = (v_i^\beta)_{i\in[n]}$ by $\Pi \circ \Pi^\beta$ and permuting vector $\boldsymbol{v}^0 = (v_i^0)_{i\in[n]}$ by Π yield the same result, as we have $\mathsf{PerVec}_\Pi(\mathsf{PerVec}_{\Pi^\beta}(\boldsymbol{v}^\beta)) = \mathsf{PerVec}_\Pi(\boldsymbol{v}^0)$. This implies $\sum_{i\in[n]} p_j^{\beta,i} v_i^\beta = \sum_{i\in[n]} p_j^{0,i} v_i^0$ for $j \in [n]$. The terms $p_j^{\beta,i} v_i^\beta = p_j^{0,i} v_i^0$ can be eliminated for all $i \in [n] \setminus \mathcal{HI}$, implying

$$\sum_{i\in\mathcal{HI}} p_j^{\beta,i} v_i^\beta = \sum_{i\in\mathcal{HI}} p_j^{0,i} v_i^0 \ \forall j \in [n]. \tag{1}$$

From the above remark, by relying on FH-IPFE and zero shares, we continue to prove that the game $\mathbf{G}_1$ in which the challenger uses $(v_i^\beta)_{i\in\mathcal{HI}}$ and $(\boldsymbol{p}^{\beta,i})_{i\in\mathcal{HI}}$ and the game $\mathbf{G}_5$ in which the challenger uses $(v_i^0)_{i\in\mathcal{HI}}$ and $(\boldsymbol{p}^{0,i})_{i\in\mathcal{HI}}$ to respond to QVote_β and $\mathsf{QPKProcess}$ queries respectively are indistinguishable.

We note that in the following games, the auxiliary key $\mathsf{ip.dk}^i_{(0,0)}$ does not help the adversary in learning the challenge bit. This is because when $\mathsf{ip.dk}^i_{(0,0)}$ is used to decrypt any honest ciphertext, the output is $[\![0]\!]_T$. For the auxiliary key $\mathsf{ip.dk}^i_{(0,1)}$, we parse the underlying vector $\boldsymbol{y}^{i,1}_{n+1}$ as $(p^{\beta,i}_{n+1} := 0, z_{n+1} := 1, 0)$. The symbols $\mathsf{share}_i^{[n]}$, $\mathsf{share}_i^{\mathcal{HI}}$, and $\mathsf{share}_{i,j}^{\mathcal{HI}}$ below denote scalars in $\mathbb{Z}_p$; their group encodings are written with brackets.

Game $\mathbf{G}_2$: The exception is that instead of generating a zero share for each $i \in \mathcal{HI}$ as $[\![\mathsf{share}_i^{[n]}]\!]_2 := t_i \cdot \mathsf{pz}_i = [\![\sum_{k\in[n]\setminus\{i\}} (-1)^{k<i} t_k t_i]\!]_2$ for any honest query on (i, v_i^0, v_i^1), the challenger generates $t_{i,k} \xleftarrow{\$} \mathbb{Z}_p$ for all $i, k \in \mathcal{HI}$ to replace values $t_k t_i$. This change enables a set of uniform zero shares between voters in $\mathcal{HI}$, each of which is denoted as $\mathsf{share}_i^{\mathcal{HI}}$ and satisfies $\mathsf{share}_i^{[n]} = \sum_{k\in[n]\setminus\mathcal{HI}} (-1)^{k<i} t_k t_i + \mathsf{share}_i^{\mathcal{HI}}$. The indistinguishability is implied by the DDH assumption in $\mathbb{G}_2$ for each instance $i, k \in \mathcal{HI}$ where, given $[\![t_i]\!]_2, [\![t_k]\!]_2$, the adversary distinguishes whether $[\![t_i t_k]\!]_2$ or $[\![t_{i,k}]\!]_2$ is given.

Game $\mathbf{G}_3$: The first exception is that, instead of using $\boldsymbol{x}_i^0 = (v_i^\beta, \mathsf{share}_i^{[n]}, 0)$ as the message for FH-IPFE encryption for any honest query on (i, v_i^0, v_i^1), the challenger uses $\boldsymbol{x}_i^1 = (0, \sum_{k\in[n]\setminus\mathcal{HI}} (-1)^{k<i} t_k t_i, 1)$.

The second exception is that, instead of using $\boldsymbol{y}_j^{i,0} = (p_j^{\beta,i}, z_j, 0)$ for all $j \in [n+1]$ for FH-IPFE functional-key generation in response to the $\mathsf{QPKProcess}$ query, the challenger uses $\boldsymbol{y}_j^{i,1} = (0, z_j, z_j \mathsf{share}_i^{\mathcal{HI}} + v_i^\beta p_j^{\beta,i})$ for all $j \in [n+1]$.

The indistinguishability is implied by the IND-CPA security of FH-IPFE, as we have $\langle \boldsymbol{x}_i^0, \boldsymbol{y}_j^{i,0} \rangle = \langle \boldsymbol{x}_i^1, \boldsymbol{y}_j^{i,1} \rangle = v_i^\beta p_j^{\beta,i} + z_j \mathsf{share}_i^{[n]}$ for $i \in \mathcal{HI}$, $j \in [n+1]$. All these vectors have dimension $\rho = 3$. The third coordinate transfers the vote contribution and the internal zero share from the ciphertext to the functional keys. The same identity covers the auxiliary key indexed by $j = n+1$, and the all-zero auxiliary key remains orthogonal to both ciphertext vectors; no fourth coordinate is needed.

Game $\mathbf{G}_4$: The exception is that, instead of using $\boldsymbol{y}_j^i = (0, z_j, z_j \mathsf{share}_i^{\mathcal{HI}} + v_i^\beta p_j^{\beta,i})$ for all $j \in [n]$ for FH-IPFE functional-key generation in response to the $\mathsf{QPKProcess}$ query, the challenger uses $\boldsymbol{y}_j^i = (0, z_j, \mathsf{share}_{i,j}^{\mathcal{HI}} + v_i^\beta p_j^{\beta,i})$ for all $j \in [n]$, where $\mathsf{share}_{i,j}^{\mathcal{HI}}$ is voter i's zero share among voters in $\mathcal{HI}$, sampled randomly for each j.

The indistinguishability is implied by the Multi-DDH assumption (given in Definition 3) in $\mathbb{G}_1$: one re-indexes $\mathcal{HI} = k_1, ..., k_{|\mathcal{HI}|}$, given random[11]

$$\{[\![\mathsf{share}_{k_i}^{\mathcal{HI}}]\!]_1\}_{i\in[|\mathcal{HI}|-1]}$$

and random $[\![z_j]\!]_1$ for each j, the adversary distinguishes whether the challenge $\{[\![c_{k_i}]\!]_1\}_{i\in[|\mathcal{HI}|-1]}$ equals $\{[\![z_j\mathsf{share}_{k_i}^{\mathcal{HI}}]\!]_1\}_{i\in[|\mathcal{HI}|-1]}$ or $\{[\![\mathsf{share}_{i,j}^{\mathcal{HI}}]\!]_1\}_{i\in[|\mathcal{HI}|-1]}$. One can simulate either $[\![z_j\mathsf{share}_{k_{|\mathcal{HI}|}}^{\mathcal{HI}}]\!]_1$ or $[\![\mathsf{share}_{k_{|\mathcal{HI}|},j}^{\mathcal{HI}}]\!]_1$ by computing

$$-\sum_{i\in[|\mathcal{HI}|-1]}[\![c_{k_i}]\!]_1.$$

Game $\mathbf{G}_5$: This game is identical to $\mathbf{G}_4$, except that, instead of using $\boldsymbol{y}_j^i = (0, z_j, \mathsf{share}_{i,j}^{\mathcal{HI}} + v_i^\beta p_j^{\beta,i})$ for all $j \in [n]$ for FH-IPFE functional-key generation in response to the QPKProcess query, the challenger uses $\boldsymbol{y}_j^i = (0, z_j, \mathsf{share}_{i,j}^{\mathcal{HI}} + v_i^0 p_j^{0,i})$. The indistinguishability is perfect: $\{\mathsf{share}_{i,j}^{\mathcal{HI}}\}_{i\in\mathcal{HI}}$ and $\{\mathsf{share}_{i,j}^{\mathcal{HI}} + (v_i^0 p_j^{0,i} - v_i^\beta p_j^{\beta,i})\}_{i\in\mathcal{HI}}$ have the same distribution for all $j \in [n]$, where $\sum_{i\in\mathcal{HI}}(v_i^0 p_j^{0,i} - v_i^\beta p_j^{\beta,i}) = 0$ by Equation (1).

In game $\mathbf{G}_5$, there is no information about the challenge bit β, which completes the proof. □

4.4 Tallying Efficiency

The protocol in Figure 8 involves the SelfTally algorithm for tallying the votes. The computational cost of this step is dominated by n^2 pairing-based FE decryptions and n^2 group additions, when all n registered users are honest and send their ballots. Although the circuit has a quadratic number of gates, its depth is logarithmic: treating pairing and group operations as unit-cost operations, the tallying computation has the structure of a matrix-vector product of dimensions $n \times n$ and n, which can be evaluated by a circuit of depth $O(\log n)$. In particular, this cost does not depend on the number of mix-servers.

5 Voting with an Offline Mix-Net

In this section, we present a more complete version of the warm-up voting protocol in Section 4, designed to support the generation of one-time pads, random permutations, and their corresponding functional encryption keys within a multi-authority framework. The enhanced protocol introduces a key modification: instead of a centralised generation of FH-IPFE secret keys, each voter generates their own secret key. While we still assume a single trusted authority for the protocol description, its role shifts significantly. During the offline processing phase, the authority now generates functional keys based on *a set of basis functional keys encrypted by voters*.

5.1 OMIX-based Voting Framework

We first provide a description of the protocol below, before making it concrete in Figure 9 and Figure 10. At a high level, preprocessing separates two tasks: OMIX hides where each voter's vote will appear, while OPAD masks the intermediate decryptions so that only their aggregate reveals the shuffled votes. Trustees complete both tasks and publish the resulting functional keys before voting. Voters then encrypt their votes, and anyone can tally using these public keys, subject to the correction requirement when voters drop out. We use "the voting framework" for the overall system and "the OMIX protocol" for its offline row-shuffling component.

KeyGen During key generation, each voter locally generates an FH-IPFE secret key along with a set of basis FH-IPFE functional keys for three-dimensional inner products. The first slot is designated for votes, the second slot for one-time pads, and the third slot is zero padding used in the security proof. The basis functional keys consist of three specific types: $(1, 0, 0)$ for decrypting the vote slot, $(0, 1, 0)$ for adding a one-time pad in the second slot, and $(0, 0, 0)$ for re-randomising the key randomness. In functional-key subscripts, we omit the final zero-padding coordinate: $\mathsf{ip.dk}_{(a,b)}^i$ denotes a key for $(a, b, 0)$. Thanks to the homomorphic property of functional keys, these basis keys can be combined to produce

[11] These shares do not appear in $\mathbb{G}_2$ in this game, even under the output of the QFaultCrt query, as voters in $\mathcal{HI}$ reveal only their pairwise randomness with dropout voters.

freshly generated functional keys of the form $(1, z_j, 0)$ or $(0, z_j, 0)$, as in the warm-up scheme. However, to prevent the combiner from directly using the key for $(1, 0, 0)$ to decrypt the vote, each voter applies a layer of public-key encryption onto that key, thereby hiding its information. Consequently, the basis functional keys must now include an encrypted version of the key for $(1, 0, 0)$.

OMIX Upon receiving the basis functional keys, the authority's role is divided into two tasks, the first of which involves shuffling the keys according to a permutation. This process begins with an identity matrix of encrypted functional keys, where the diagonal elements correspond to the $(1, 0, 0)$ keys of each voter. Each column is associated with a voter, while the remaining entries in each column are filled with $(0, 0, 0)$ keys that are parsed as encrypted with null randomness. The authority then permutes the rows of this matrix and re-randomises the public-key encryption layer of each functional key to ensure that the applied permutation remains hidden.

OPAD The second task is to embed one-time pads and functional-key randomness into the functional keys, producing an output also structured as a matrix. This step leverages the homomorphic properties of functional encryption: for each matrix entry, the authority multiplies the $(0, 1, 0)$ keys by the one-time pads and the $(0, 0, 0)$ keys by the randomness, then sums the results encrypted under the public key, ensuring that the embedded one-time pads and randomness remain hidden before they are added to keys from OMIX. To collaboratively generate the one-time pads $[\![z]\!]_1$ in a multi-authority setting, we additionally require that they be encrypted to ensure independence.

To finalise the offline processing phase, the authority sums the matrices of public-key ciphertexts obtained from the previous tasks and performs a decryption. Due to the structure-preserving properties of functional keys under the homomorphic public-key encryption scheme, the decrypted functional keys are identical to those that would have been generated in the warm-up scheme.

Modifying the warm-up protocol into the above one may pose a vulnerability to copy-ballot attacks. In this attack, an adversary can target a voter by submitting an identical set of basis functional keys and later casting an identical ciphertext as its ballot. This succeeds because the processed functional keys corresponding to the adversary can then decrypt the copied ballot. To prevent this, we impose non-malleability on basis functional keys by requiring each voter to provide a proof of knowledge for the vote-slot decrypting key $(1, 0)$, i.e. the plaintext under the public-key encryption. Adapted from the idea in [Gro04b], this proof is constructed using a non-interactive Σ-protocol, which is hardwired with the voter's identity.

The building blocks to ensure ballot privacy of the voting protocol in Figure 9 will contain an FH-IPFE scheme ($\mathsf{ip.Setup}, \mathsf{ip.DKGen}$, $\mathsf{ip.Enc}, \mathsf{ip.Dec}$), a PKE scheme ($\mathsf{PKE.SetUp}, \mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$), and a Σ-protocol denoted by $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ (described in Figure 11). Furthermore, in order to complete the protocol with universal verifiability, we will also use NIZK proof systems to ensure correct execution of parties, namely for the following relations: $\mathcal{R}_{\mathsf{dk}}$ for voter public keys, $\mathcal{R}_{\mathsf{ct}}$ for voter encrypted ballots, $\mathcal{R}_{\mathsf{Mix}}$ for shuffling functional keys, $\mathcal{R}_{\mathsf{otp}}$ for generating one-time pads and key randomness, $\mathcal{R}_{\mathsf{Dec}}$ for decrypting in the offline phase, and $\mathcal{R}_{\mathsf{ck}}$ for generating correcting keys in case of dropout voters.

We prove correctness, corrective fault tolerance, and client-side lightweightness in Section 5.2, universal verifiability in Section 5.3, and ballot privacy in Section 5.4. Ballot privacy is proved under the assumption that the OMIX process and the OPAD process are executed by a trusted server. While oracle access to these protocols is sufficient here, we introduce concrete constructions of OMIX and OPAD in Section 6.4 and Section 7.3, respectively, which account for the presence of malicious servers. The security integration of these two offline protocols into the voting framework is given in Section 7.4.

5.2 Correctness, Corrective Fault Tolerance, and Client-Side Lightweightness

Correctness. On n registered public keys $\{\mathsf{pk}_i\}_{i\in[n]} = \{\mathsf{id}_i, \mathsf{ct}_i, \mathsf{ip.dk}^i_{(0,1)}, \mathsf{ip.dk}^i_{(0,0)}, \pi_i^{\mathsf{dk}}, \pi_i^{\mathsf{FS}}, [\![t_i]\!]_2\}_{i\in[n]}$, the $\mathsf{PKProcess}$ algorithm outputs MBox such that[12]

$$\mathsf{MBox}(j,i) = \begin{cases} \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(1,0;\gamma_i^{\mathsf{ip.dk}})}; \gamma_{\mathsf{PKE},j,i}) & \text{if } j = \Pi(i), \\ \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(0,0;\gamma_i'^{\mathsf{ip.dk}})}; \gamma_{\mathsf{PKE},j,i}) & \text{if } j \neq \Pi(i). \end{cases}$$

[12] We denote by $\gamma_i^{\mathsf{ip.dk}}$, $\gamma_i'^{\mathsf{ip.dk}}$, and $\gamma_{\mathsf{IPE},j,i}$ the randomness used to generate the FE keys.

$\underline{\mathsf{SetUp}(1^\lambda)}$: The authority specifies $\mathbb{V} = \{1, \dots V\}$ and generates $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.SetUp}(1^\lambda)$ and $\mathsf{NIZK.pp} \leftarrow \mathsf{NIZK}_{\{\mathcal{R}_{\mathsf{dk}}, \mathcal{R}_{\mathsf{FS}}, \mathcal{R}_{\mathsf{ct}}, \mathcal{R}_{\mathsf{Mix}}, \mathcal{R}_{\mathsf{otp}}, \mathcal{R}_{\mathsf{Dec}}, \mathcal{R}_{\mathsf{ck}}\}}.\mathsf{SetUp}(1^\lambda)$. It outputs $\mathsf{mpk} = (\mathbb{V}, \mathsf{NIZK.pp}, \mathsf{PKE.pk}, \mathcal{PG})$ and stores $\mathsf{msk} = \mathsf{PKE.sk}$.

Apart from SetUp, the offline phase is detailed in Figure 10:

- Each voter id computes $(\mathsf{pk}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{id}, \mathsf{mpk})$[a].
- The authority runs $\mathsf{Register}(\mathsf{L}, \mathsf{pk}_{\mathsf{id}})$ to update L with $\mathsf{pk}_{\mathsf{id}}$.
- The authority outputs $\mathsf{hpk} \leftarrow \mathsf{PKProcess}(\mathsf{L}, \mathsf{msk})$.

$\underline{\mathsf{Vote}(\mathsf{ek}_{\mathsf{id}}, v, \mathsf{hpk})}$: Each voter id parses $\mathsf{ek}_{\mathsf{id}} = (\mathsf{pk}_{\mathsf{id}}, \mathsf{ip.sk}_{\mathsf{id}}, t_{\mathsf{id}})$ and computes

- an FH-IPFE encryption: $\mathsf{ip.ct}^{\mathsf{id}} \leftarrow \mathsf{ip.Enc}(\mathsf{ip.sk}_{\mathsf{id}}, ([\![v]\!]_2, t_{\mathsf{id}} \cdot \mathsf{pz}_{\mathsf{id}}, [\![0]\!]_2))$;
- a proof for correct ciphertext: $\pi^{\mathsf{ct}}_{\mathsf{id}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ct}}}.\mathsf{Prove}(\mathsf{ip.ct}^{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}}, \mathsf{pz}_{\mathsf{id}}; \mathsf{ip.sk}_{\mathsf{id}}, t_{\mathsf{id}}, v)$;

and casts $b_{\mathsf{id}} = (\mathsf{ip.ct}^{\mathsf{id}}, \pi^{\mathsf{ct}}_{\mathsf{id}})$ to BB. A registered voter id can abort voting.

$\underline{\mathsf{Append}(\mathsf{BB}, b_{\mathsf{id}}, \mathsf{L}, \mathsf{hpk})}$: Parsing $b_{\mathsf{id}} = (\mathsf{ip.ct}^{\mathsf{id}}, \pi^{\mathsf{ct}}_{\mathsf{id}})$, the authority appends b_{id} to BB if

$$\mathsf{id} \text{ is in } \mathsf{L} \text{ and there does not exist } b'_{\mathsf{id}} \in \mathsf{BB} \text{ and } \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ct}}}.\mathsf{Verify}(\pi^{\mathsf{ct}}_{\mathsf{id}}) = 1;$$

and rejects otherwise.

$\underline{\mathsf{FaultAgg}(\mathsf{BB}, \mathsf{hpk}, \mathsf{id})}$: It parses $\mathsf{V} \subseteq [n]$ as the list of voters id in L whose ballots are in BB. If $\mathsf{V} = [n]$, nothing is output. Otherwise, the fault aggregate for the voter $i \in \mathsf{V}$ that corresponds to id is $\mathsf{fa}_i = \sum_{k \in [n] \setminus \mathsf{V}} (-1)^{k<i} \cdot [\![t_k]\!]_2$.

$\underline{\mathsf{FaultCrt}(\mathsf{fa}_{\mathsf{id}}, \mathsf{ek}_{\mathsf{id}})}$: The correcting key is output as $\mathsf{ck}_{\mathsf{id}} = t_{\mathsf{id}} \cdot \mathsf{fa}_{\mathsf{id}}$. It generates a proof for correcting key $\pi^{\mathsf{id}}_{\mathsf{ck}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ck}}}.\mathsf{Prove}(\mathsf{fa}_{\mathsf{id}}, [\![t_{\mathsf{id}}]\!]_2, \mathsf{ck}_{\mathsf{id}}; t_{\mathsf{id}})$ and outputs $(\mathsf{ck}_{\mathsf{id}}, \pi^{\mathsf{id}}_{\mathsf{ck}})$.

$\underline{\mathsf{SelfTally}(\mathsf{BB}, \{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}, \mathsf{hpk})}$: Let $\mathsf{V} \subseteq [n]$ be the list of each voter $\mathsf{id}_i \in \mathsf{L}$ whose ballot is in BB and for which ck_i is given (in case $\mathsf{V} \subsetneq [n]$) and let $\boldsymbol{v}^\star$ be an empty tuple. It obtains $\{\mathsf{ip.dk}^i_j\}_{j,i \in [n]}$ from hpk, $\{\mathsf{ip.ct}^i\}_{i \in \mathsf{V}}$ from BB. For each $j \in [n]$, it appends v'_j from $[\![v'_j]\!]_T = \sum_{i \in \mathsf{V}} \mathsf{ip.Dec}(\mathsf{ip.dk}^i_j, \mathsf{ip.ct}^i) - e([\![z_j]\!]_1, \sum_{i \in \mathsf{V}} \mathsf{ck}_i)$ to $\boldsymbol{v}^\star$ if $[\![v'_j]\!]_T \neq [\![0]\!]_T$ and v'_j can be recovered. The result res is the equivalence class $[\boldsymbol{v}^\star]_\sim$.

[a] The argument msk is not required in this construction.

Fig. 9: OMIX-based voting framework

$\underline{\mathsf{Verify}(\mathsf{BB}, \{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}, \mathsf{res}, \mathsf{L}, \mathsf{hpk})}$: It parses $\mathsf{L} = (\mathsf{pk}_i)_{i \in [n]}$ and $\mathsf{V} \subseteq [n]$ as the list of each voter $\mathsf{id}_i \in \mathsf{L}$ whose ballot is in BB and for whom ck_i is given (in case $\mathsf{V} \subsetneq [n]$). Let L_i (and BB_i) be the state of L (and BB) before pk_i (and b_i) is appended. It verifies the following phases:

- registering: it verifies whether $\mathsf{Register}(\mathsf{L}_i, \mathsf{pk}_i)$ outputs 1 for $i \in [n]$;
- processing: it verifies π^{Proc} and $\mathsf{pz}_i = \sum_{k \in [n] \setminus \{i\}} (-1)^{k<i} \cdot [\![t_k]\!]_2$ for $i \in [n]$ in hpk;
- ballot appending: it verifies whether $\mathsf{Append}(\mathsf{BB}_i, b_i, \mathsf{L}, \mathsf{hpk})$ outputs 1 for $i \in \mathsf{V}$;
- fault correction: it computes $\mathsf{fa}_i = \mathsf{FaultAgg}(\mathsf{BB}, \mathsf{hpk}, \mathsf{id})$ and verifies π^i_{ck} for $i \in \mathsf{V}$;
- tallying: it verifies whether $\mathsf{res} = \mathsf{SelfTally}(\mathsf{BB}, \{\mathsf{ck}_{\mathsf{id}}\}_{\mathsf{id}}, \mathsf{hpk})$;

and outputs 1 if all the phases are valid, and 0 otherwise.

Fig. 9: OMIX-based voting framework (continued): public verification.

KeyGen(id, mpk): Each voter id proceeds as follows

1. It generates a zero-sharing secret key $t_{\mathsf{id}} \xleftarrow{\$} \mathbb{Z}_p$.
2. It generates an FH-IPFE secret key $\mathsf{ip.sk}_{\mathsf{id}} \leftarrow \mathsf{ip.Setup}(1^\lambda, 1^3)$ and functional keys
   (a) for vote-slot decrypting: $\mathsf{ip.dk}^{\mathsf{id}}_{(1,0)} \leftarrow \mathsf{ip.DKGen}(\mathsf{ip.sk}_{\mathsf{id}}, [\![1,0,0]\!]_1)$;
   (b) for one-time-pad adding: $\mathsf{ip.dk}^{\mathsf{id}}_{(0,1)} \leftarrow \mathsf{ip.DKGen}(\mathsf{ip.sk}_{\mathsf{id}}, [\![0,1,0]\!]_1)$;
   (c) for re-randomising: $\mathsf{ip.dk}^{\mathsf{id}}_{(0,0)} \leftarrow \mathsf{ip.DKGen}(\mathsf{ip.sk}_{\mathsf{id}}, [\![0,0,0]\!]_1)$.
3. It encrypts the functional key for vote-slot decrypting under PKE as

$$\mathsf{ct}_{\mathsf{id}} \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^{\mathsf{id}}_{(1,0)});$$
$$\pi^{\mathsf{FS}}_{\mathsf{id}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{Prove}(\mathsf{id}, \mathsf{PKE.pk}, \mathsf{ct}_{\mathsf{id}}; \mathsf{ip.dk}^{\mathsf{id}}_{(1,0)}).$$

4. It generates $\pi^{\mathsf{dk}}_{\mathsf{id}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{dk}}}.\mathsf{Prove}(\mathsf{ip.dk}^{\mathsf{id}}_{(0,1)}, \mathsf{ip.dk}^{\mathsf{id}}_{(0,0)}, \mathsf{ct}_{\mathsf{id}}; \mathsf{ip.sk}_{\mathsf{id}})$ for the correct generation of (encrypted) functional keys.
5. It outputs $\mathsf{pk}_{\mathsf{id}} = \{\mathsf{id}, \mathsf{ct}_{\mathsf{id}}, \mathsf{ip.dk}^{\mathsf{id}}_{(0,1)}, \mathsf{ip.dk}^{\mathsf{id}}_{(0,0)}, \pi^{\mathsf{dk}}_{\mathsf{id}}, \pi^{\mathsf{FS}}_{\mathsf{id}}, [\![t_{\mathsf{id}}]\!]_2\}$ and stores $\mathsf{ek}_{\mathsf{id}} = (\mathsf{pk}_{\mathsf{id}}, \mathsf{ip.sk}_{\mathsf{id}}, t_{\mathsf{id}})$.

Register(L, $\mathsf{pk}_{\mathsf{id}}$): Parsing $\mathsf{pk}_{\mathsf{id}} = \{\mathsf{id}, \mathsf{ct}_{\mathsf{id}}, \mathsf{ip.dk}^{\mathsf{id}}_{(0,1)}, \mathsf{ip.dk}^{\mathsf{id}}_{(0,0)}, \pi^{\mathsf{dk}}_{\mathsf{id}}, \pi^{\mathsf{FS}}_{\mathsf{id}}, [\![t_{\mathsf{id}}]\!]_2\}$, the authority appends $\mathsf{pk}_{\mathsf{id}}$ to L if there does not exist $\mathsf{pk}'_{\mathsf{id}} \in \mathsf{L}$ and $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{Verify}(\pi^{\mathsf{FS}}_{\mathsf{id}}) = 1$ and $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{dk}}}.\mathsf{Verify}(\pi^{\mathsf{dk}}_{\mathsf{id}}) = 1$.

Fig. 10: Description of offline-phase algorithms in Figure 9.

PKProcess(L, msk): The authority proceeds as follows:

1. It parses $\mathsf{L} = (\mathsf{pk}_i)_{i\in[n]} = (\mathsf{id}_i, \mathsf{ct}_i, \mathsf{ip.dk}^i_{(0,1)}, \mathsf{ip.dk}^i_{(0,0)}, \pi^{\mathsf{dk}}_i, \pi^{\mathsf{FS}}_i, [\![t_i]\!]_2)_{i\in[n]}$.
2. For $i \in [n]$, it computes a zero-sharing public key: $\mathsf{pz}_i = \sum_{k\in[n]\setminus\{i\}} (-1)^{k<i} \cdot [\![t_k]\!]_2$
3. It generates a random $\Pi : [n] \to [n]$, $\boldsymbol{z} = (z_j)_{j\in[n]}$ and $(\gamma_{\mathsf{IPE},j,i})_{j,i\in[n]}$.
4. The initial $n \times n$-mix box is parsed by default as MBox_I where

$$\mathsf{MBox}_I(j,i) = \begin{cases} \mathsf{ct}_i & \text{if } j = i, \\ \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(0,0)}; 0) & \text{if } j \neq i. \end{cases}$$

5. OMIX: It generates random $(\gamma_{\mathsf{PKE},j,i})_{j,i\in[n]}$ and mixes MBox_I into MBox as

$$\mathsf{MBox}(\Pi(j), i) = \mathsf{MBox}_I(j,i) + \gamma_{\mathsf{PKE},j,i} \cdot \mathsf{PKE.Enc}(\mathsf{PKE.pk}, [\![\mathbf{0}]\!]_1; 1).$$

6. OPAD: It generates $\mathsf{ct}^{\mathsf{PAD}}_j = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, [\![z_j]\!]_1)$ and OBox as

$$\mathsf{OBox}(j,i) = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, z_j \cdot \mathsf{ip.dk}^i_{(0,1)} + \gamma_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)}).$$

7. The authority decrypts $\mathsf{ct}^{\mathsf{PAD}}_j$ into $[\![z_j]\!]_1$ and obtains the functional keys as

$$\mathsf{ip.dk}^i_j = \mathsf{PKE.Dec}(\mathsf{PKE.sk}, \mathsf{OBox}(j,i) + \mathsf{MBox}(j,i)) \ \forall j, i \in [n]$$

8. It generates a proof for correct processing as π^{Proc}, consisting of

$$\pi^{\mathsf{Mix}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{Mix}}}.\mathsf{Prove}(\mathsf{MBox}, (\mathsf{pk}_i)_{i\in[n]}, \mathsf{PKE.pk}; \Pi, (\gamma_{\mathsf{PKE},j,i})_{j,i\in[n]})$$
$$\pi^{\mathsf{otp}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{otp}}}.\mathsf{Prove}\big(\mathsf{OBox}, (\mathsf{ct}^{\mathsf{PAD}}_j)_{j\in[n]}, (\mathsf{pk}_i)_{i\in[n]}, \mathsf{PKE.pk}; (z_j)_{j\in[n]}, (\gamma_{\mathsf{IPE},j,i})_{j,i\in[n]}\big)$$
$$\pi^{\mathsf{Dec}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{Dec}}}.\mathsf{Prove}([\![\boldsymbol{z}]\!]_1, (\mathsf{ip.dk}^i_j)_{j,i\in[n]}, (\mathsf{ct}^{\mathsf{PAD}}_j)_{j\in[n]}, \mathsf{MBox}, \mathsf{OBox}; \mathsf{PKE.sk})$$

9. It publishes

$$\mathsf{hpk} = \mathsf{mpk} || (\mathsf{ct}^{\mathsf{PAD}}_j)_{j\in[n]}, \mathsf{MBox}, \mathsf{OBox}, (\mathsf{ip.dk}^i_j)_{j,i\in[n]}, \pi^{\mathsf{Proc}}, \{\mathsf{pz}_i\}_{i\in[n]}, [\![\boldsymbol{z}]\!]_1$$

Fig. 10: Offline-phase algorithms (continued): public-key preprocessing.

and OBox such that

$$\mathsf{OBox}(j,i) = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(0,z_j;\gamma_{\mathsf{IPE},j,i})}; \gamma^{\mathsf{OBox}}_{j,i}) \ \forall i,j \in [n]$$

thanks to the structure-preserving homomorphic property of PKE ciphertexts and FE keys. Furthermore, one has

$$\begin{aligned}&\mathsf{MBox}(j,i) + \mathsf{OBox}(j,i)\\ &= \begin{cases}\mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(1,z_j;\gamma_i^{\mathsf{ip.dk}}+\gamma_{\mathsf{IPE},j,i})}; \gamma_{\mathsf{PKE},j,i} + \gamma^{\mathsf{OBox}}_{j,i}) & \text{if } j = \Pi(i),\\ \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(0,z_j;\gamma_i'^{\mathsf{ip.dk}}+\gamma_{\mathsf{IPE},j,i})}; \gamma_{\mathsf{PKE},j,i} + \gamma^{\mathsf{OBox}}_{j,i}) & \text{if } j \neq \Pi(i).\end{cases}\end{aligned}$$

Therefore, the functional decryption keys output from the $\mathsf{PKProcess}$ algorithm will be in the same form as in the warm-up voting protocol, i.e.

$$\mathsf{ip.dk}^i_j = \mathsf{ip.DKGen}(\mathsf{ip.sk}_i, [\![p^i_j, z_j, 0]\!]_1)$$

where the third coordinate is zero padding used in the security proof and $\boldsymbol{p}^i$ is a vector that represents $\Pi(i)$ such that $p^i_j = 1$ if $j = \Pi(i)$ and $p^i_j = 0$ otherwise. Furthermore, since the vote encryptions $(\mathsf{ip.ct}^i)_{i\in[n]}$ are the same as in the warm-up voting protocol, the decryption by $\sum_{i\in[n]} \mathsf{ip.Dec}(\mathsf{ip.dk}^i_j, \mathsf{ip.ct}^i)\ \forall j \in [n]$ is also correct. The correctness of the protocol is complete, as all proofs $\{\pi^{\mathsf{ct}}_i, \pi^{\mathsf{dk}}_i, \pi^{\mathsf{FS}}_i\}_{i\in[n]}, \pi^{\mathsf{Mix}}, \pi^{\mathsf{otp}}, \pi^{\mathsf{Dec}}$, and $\{\pi^i_{\mathsf{ck}}\}_{i\in[n]}$ are valid due to the completeness of the respective proof systems.

Corrective Fault Tolerance. The corrective fault tolerance is achieved as in the warm-up protocol and can be verified by inspection.

Client-Side Lightweightness. The protocol guarantees that both the per-voter communication and computation costs remain constant with respect to the total number of voters. Since the number of operations in each FE algorithm is $O_\lambda(1)$, the corresponding proofs for correct ciphertexts and functional keys also have complexities in $O_\lambda(1)$. Specifically, we have:

- In $\mathsf{KeyGen}(\mathsf{id}, \mathsf{mpk})$, each voter generates one element in $\mathbb{G}_2$, one FE secret key, three functional keys in $\mathbb{G}_1$, one public-key encryption in $\mathbb{G}_1$, and two corresponding correctness proofs for the functional keys and the public-key encryption.
- In $\mathsf{Vote}(\mathsf{ek_{id}}, v, \mathsf{hpk})$, each voter generates one FE ciphertext in $\mathbb{G}_2$ and a correctness proof for its generation.
- In $\mathsf{FaultCrt}(\mathsf{fa_{id}}, \mathsf{ek_{id}})$, each voter generates one element in $\mathbb{G}_2$ and a proof of knowledge of a discrete logarithm.

Hence, the protocol satisfies client-side lightweightness as defined in Definition 13.

5.3 Universal Verifiability

Theorem 2 (Universal Verifiability). *The voting protocol described in Figure 9 and Figure 10 achieves universal verifiability as per Definition 15, assuming the soundness property of the NIZK proof systems for the relations $\mathcal{R}_{\mathsf{dk}}$, $\mathcal{R}_{\mathsf{ct}}$, $\mathcal{R}_{\mathsf{Mix}}$, $\mathcal{R}_{\mathsf{otp}}$, $\mathcal{R}_{\mathsf{Dec}}$ and $\mathcal{R}_{\mathsf{ck}}$.*

Proof. Assume that at the end of the soundness game described in Definition 15, one has $\mathsf{Verify}(\mathsf{BB} := \{b_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{HS}\cup\mathcal{CS}}, \{\mathsf{ck_{id}}\}_{\mathsf{id}\in\mathcal{HS}\cup\mathcal{CS}}, \mathsf{res}, \mathsf{L} := \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{hpk}) = 1$, where $\mathcal{HS}$ is the set of honest voters and $\mathcal{CS}$ is the set of corrupted voters. For this protocol, successful verification implies that $(\mathsf{pk_{id}})_{\mathsf{id}\in\mathcal{HS}\cup\mathcal{CS}}$ are contained in the list $\mathsf{L} := \{\mathsf{pk}_i\}_{i\in[n]}$ of some n voters. The following verifications are also valid:

- Registering: the validity of π^{dk}_i for all $i \in [n]$ implies that the basis keys $\{\mathsf{ct}_i, \mathsf{ip.dk}^i_{(0,1)}, \mathsf{ip.dk}^i_{(0,0)}, [\![t_i]\!]_2\}_{i\in[n]}$ are generated as in the KeyGen algorithm.
- Public-key processing: the validity of π^{Mix} implies that the mixing process is valid, i.e. the MBOX is generated from the basis functional keys of all voters in L; the validity of π^{otp} implies that the process of generating one-time pads and key randomness is valid, i.e. the terms in OBox and $(\mathsf{ct}^{\mathsf{PAD}}_j)_{j\in[n]}$ are consistently and correctly generated. Lastly, the validity of π^{Dec} implies that $(\mathsf{ct}^{\mathsf{PAD}}_j)_{j\in[n]}$ and the sum of MBOX and OBox are correctly decrypted, yielding the correct $[\![\boldsymbol{z}]\!]_1$ and decryption keys $(\mathsf{ip.dk}^i_j)_{j,i\in[n]}$, respectively, as in the $\mathsf{PKProcess}$ algorithm.

- Ballot appending: the validity of π_i^{ct} for all $i \in \mathcal{CS}$ implies that the ciphertexts $(\mathsf{ip.ct}^i)_{i\in\mathcal{CS}}$ are generated correctly with respect to the FE secret keys $(\mathsf{ip.sk}_i)_{i\in\mathcal{CS}}$ used in the basis functional keys $\{\mathsf{ct}_i, \mathsf{ip.dk}^i_{(0,1)}, \mathsf{ip.dk}^i_{(0,0)}\}_{i\in\mathcal{CS}}$ in L.
- Fault correction: the validity of π^i_{ck} for all $i \in \mathcal{CS}$ implies that the correcting keys ck_i are generated correctly as in the $\mathsf{FaultCrt}$ algorithm, with respect to the registered keys $[\![t_i]\!]_2$ and the fault aggregates fa_i for the set $n \setminus (\mathcal{HS} \cup \mathcal{CS})$.
- Tallying: the validity of res implies that it is equal to the output of the public algorithm $\mathsf{SelfTally}(\mathsf{BB}, \{\mathsf{ck}_i\}_{i\in\mathcal{CS}\cup\mathcal{HS}}, \mathsf{hpk})$, which by the corrective-fault-tolerance property returns a permutation of $\{v_i\}_{i\in\mathcal{HS}\cup\mathcal{CS}'}$ where $\{v_i\}_{i\in\mathcal{HS}}$ are valid votes chosen by the adversary and $\{v_i\}_{i\in\mathcal{CS}'}$ are valid votes that can be recovered from the set of corrupted voters' ballots.

The protocol therefore satisfies universal verifiability. □

5.4 Ballot Privacy

We first provide in Figure 11 a specification of the non-interactive Σ-protocol, denoted as $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$, that is used to avoid copy-ballot attacks in the voting protocol with an offline mix-net (see Figure 9 and Figure 10). Both the soundness and zero-knowledge properties of $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ rely on the random oracle for rewinding and programming, respectively.

- *Knowledge Soundness*: Given two transcripts $(R_{\mathsf{FS}}, \boldsymbol{R}_{\mathsf{mess}}, e, \boldsymbol{\rho}_{\mathsf{mess}}, \rho_{\mathsf{FS}})$ and $(R_{\mathsf{FS}}, \boldsymbol{R}_{\mathsf{mess}}, e', \boldsymbol{\rho}'_{\mathsf{mess}}, \rho'_{\mathsf{FS}})$ as input, the extractor will output $\boldsymbol{m} := (\boldsymbol{\rho}_{\mathsf{mess}} - \boldsymbol{\rho}'_{\mathsf{mess}})/(e - e')$ and $r := (\rho_{\mathsf{FS}} - \rho'_{\mathsf{FS}})/(e - e')$ as witness.
- *Zero-Knowledge Property*: Given a statement $(\mathsf{id}, \mathsf{pk} := \boldsymbol{P}, \mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2))$ as input, the zero-knowledge simulator will output a proof $\pi_{\mathsf{FS}} = (\mathsf{id}, \mathsf{pk}, \mathsf{ct}, R_{\mathsf{FS}} := [\![\rho_{\mathsf{FS}}]\!] - e \cdot \mathsf{ct}_2, \boldsymbol{R}_{\mathsf{mess}} := \rho_{\mathsf{FS}} \cdot \boldsymbol{P} + [\![\boldsymbol{\rho}_{\mathsf{mess}}]\!] - e \cdot \mathsf{ct}_1, \boldsymbol{\rho}_{\mathsf{mess}}, \rho_{\mathsf{FS}})$ where $\boldsymbol{\rho}_{\mathsf{mess}}, \rho_{\mathsf{FS}}$ and e are selected at random and set $\mathcal{H}(\mathsf{id}, \mathsf{pk}, \mathsf{ct}, R_{\mathsf{FS}}, \boldsymbol{R}_{\mathsf{mess}}) := e$.

$\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{SetUp}(1^\lambda)$: Outputs a group $(\mathbb{G}, p)$ and a hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$

$\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{Prove}(\mathsf{id}, \mathsf{pk} := \boldsymbol{P}, \mathsf{ct} := (r \cdot \boldsymbol{P} + [\![\boldsymbol{m}]\!], [\![r]\!]); (\boldsymbol{m}, r))$:

1. Commit randomness $r_{\mathsf{FS}} \xleftarrow{\$} \mathbb{Z}_p$ as $R_{\mathsf{FS}} = [\![r_{\mathsf{FS}}]\!]$ and $\boldsymbol{r}_{\mathsf{mess}} \xleftarrow{\$} \mathbb{Z}_p^{|\boldsymbol{m}|}$ as $\boldsymbol{R}_{\mathsf{mess}} = r_{\mathsf{FS}} \cdot \boldsymbol{P} + [\![\boldsymbol{r}_{\mathsf{mess}}]\!]$;
2. Compute a challenge $e \leftarrow \mathcal{H}(\mathsf{id}, \mathsf{pk}, \mathsf{ct}, R_{\mathsf{FS}}, \boldsymbol{R}_{\mathsf{mess}})$;
3. Compute a response $(\boldsymbol{\rho}_{\mathsf{mess}} = \boldsymbol{r}_{\mathsf{mess}} + e\boldsymbol{m}, \rho_{\mathsf{FS}} = r_{\mathsf{FS}} + er)$ and outputs

$$\pi_{\mathsf{FS}} = (\mathsf{id}, \mathsf{pk}, \mathsf{ct}, R_{\mathsf{FS}}, \boldsymbol{R}_{\mathsf{mess}}, \boldsymbol{\rho}_{\mathsf{mess}}, \rho_{\mathsf{FS}});$$

$\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{Verify}(\pi_{\mathsf{FS}} := (\mathsf{id}, \mathsf{pk}, \mathsf{ct}, R_{\mathsf{FS}}, \boldsymbol{R}_{\mathsf{mess}}, \boldsymbol{\rho}_{\mathsf{mess}}, \rho_{\mathsf{FS}}))$:

1. Compute the challenge $e \leftarrow \mathcal{H}(\mathsf{id}, \mathsf{pk}, \mathsf{ct}, R_{\mathsf{FS}}, \boldsymbol{R}_{\mathsf{mess}})$;
2. Parse $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$;
3. Verify if $e \cdot \mathsf{ct}_1 + \boldsymbol{R}_{\mathsf{mess}} = \rho_{\mathsf{FS}} \cdot \boldsymbol{P} + [\![\boldsymbol{\rho}_{\mathsf{mess}}]\!]$ and $e \cdot \mathsf{ct}_2 + R_{\mathsf{FS}} = [\![\rho_{\mathsf{FS}}]\!]$;
4. If yes, output 1, and 0 otherwise.

Fig. 11: The $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ protocol for ElGamal encryption used in Section 5

Theorem 3 (Ballot Privacy). *The voting protocol described in Figure 9 and Figure 10 achieves ballot privacy in the selective and symmetric-key setting (as per Definition 14), assuming the zero-knowledge property of the NIZK proof systems for the relations in* $\{\mathcal{R}_{\mathsf{dk}}, \mathcal{R}_{\mathsf{ct}}, \mathcal{R}_{\mathsf{Mix}}, \mathcal{R}_{\mathsf{otp}}, \mathcal{R}_{\mathsf{Dec}}, \mathcal{R}_{\mathsf{ck}}\}$*, the random oracle in the* $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ *proof system (described in Figure 11), the IND-CPA security of the* PKE *scheme, the SXDH assumption in* $\mathcal{PG}$ *and the IND-CPA security of the* FH-IPFE *scheme.*

Proof. At the end of the game, let L be the list of eligible voters, $\mathcal{CS}$ be the set of voters that are corrupted via QCor, and $\mathcal{HS}$ be the subset of honest voters (whose public keys are generated via $\mathsf{QNewHon}$ but not corrupted via QCor) that are queried and recorded in QVote_β. In the selective setting, $\mathcal{HS}$, $\mathcal{CS}$, and $\{(\mathsf{id}, v^0, v^1)\}_{\mathsf{id}\in\mathcal{HS}}$ are specified beforehand from the $\mathsf{QNewHon}$, QCor and QVote_β queries. In the symmetric-key setting, any (id, v^0, v^1) vote queries for $\mathsf{id} \in \mathcal{CS}$ must satisfy $v^0 = v^1$.

The list L also contains honest dropout voters with no recorded QVote_β queries, denoted by $\mathcal{DH}$, and voters registered by the adversary via $\mathsf{QReqCast}$ queries, denoted by $\mathcal{RC}$. The four sets $\mathcal{HS}$, $\mathcal{CS}$, $\mathcal{DH}$, and $\mathcal{RC}$ partition L.

To align with permutations, we assume $n = |\mathsf{L}|$ and index the voters in L using integers from $[n]$. From this point forward, we denote by $\mathcal{HI}$, $\mathcal{CI}$, $\mathcal{DI}$, and $\mathcal{RI}$ the sets of indices in $[n]$ corresponding to the voters in $\mathcal{HS}$, $\mathcal{CS}$, $\mathcal{DH}$, and $\mathcal{RC}$, respectively.

Compared to the warm-up scheme in Figure 8, the protocol in Figure 9 exposes more information about secrets of voters in $\mathcal{HI}$, whose ballots contain the challenge bit. Besides zero-knowledge proofs, the adversary also gains additional access to:

- the functional keys $\mathsf{ip.dk}^i_{(1,0)}$ for decrypting the vote slot, which are encrypted under PKE in ct_i of pk_i;
- the functional keys embedding $\{\Pi(i)\}_{i\in\mathcal{HI}}$ that are encrypted under PKE as columns $\{i\}_{i\in\mathcal{HI}}$ in MBox of hpk;
- the functional keys embedding the one-time pads $\{z_j\}_{j\in[n]}$ and the randomness $\{\gamma_{\mathsf{IPE},j,i}\}_{j\in[n],i\in\mathcal{HI}}$, encrypted under PKE as columns $\{i\}_{i\in\mathcal{HI}}$ in OBox of hpk.

To eliminate this additional leakage, we modify the protocol so that all the above PKE ciphertexts instead encrypt a null value $\mathbf{0}$. More formally, we introduce the following hybrid games to transition from the real game $\mathbf{G}_0$ to the game $\mathbf{G}_3$, where the adversary's view contains no more information on secrets of voters in $\mathcal{HI}$ than in the warm-up scheme.

Game $\mathbf{G}_1$: In this game, via a hybrid, we switch all the proof systems that ensure universal verifiability, namely those for the relations in $\{\mathcal{R}_{\mathsf{dk}}, \mathcal{R}_{\mathsf{ct}}, \mathcal{R}_{\mathsf{Mix}}, \mathcal{R}_{\mathsf{otp}}, \mathcal{R}_{\mathsf{Dec}}, \mathcal{R}_{\mathsf{ck}}\}$, to the zero-knowledge setting by using a simulator to generate proofs[13]. Through a hybrid argument, one can use the adversary $\mathcal{A}$ to construct a distinguisher between two settings for one of the proof systems.

Game $\mathbf{G}_2$: In this game, by relying on the programmability of the random oracle $\mathcal{RO}$, the challenger simulates Σ-protocol proofs π_i^{FS} for knowledge of the plaintexts under ciphertexts ct_i that are generated via $\mathsf{QNewHonest}$ queries. Let $(\mathsf{PKE.pk}, \mathsf{ct}_i)$ be the statement and id_i the auxiliary information; the challenger chooses z as a response, e as a random challenge, and computes the initial message a. Provided that $(\mathsf{id}_i, (\mathsf{PKE.pk}, \mathsf{ct}_i), a)$ has not been queried before by the adversary, the challenger assigns e as the answer to the query $\mathcal{RO}(\mathsf{id}_i, (\mathsf{PKE.pk}, \mathsf{ct}_i), a)$. The indistinguishability between $\mathbf{G}_1$ and $\mathbf{G}_2$ is statistical, as the probability that the adversary correctly guesses the randomness in ct_i, which is generated honestly in the $\mathsf{QNewHonest}$ oracle, and queries it to $\mathcal{RO}$ is negligible.

Game $\mathbf{G}_3$: In this game, the challenger replaces PKE ciphertexts by the following strategy: it first replaces the ciphertexts ct_i that are queried via $\mathsf{QNewHonest}$ by encryptions of $\mathbf{0}$. It then runs the $\mathsf{PKProcess}$ algorithm on the replaced ciphertexts ct_i (in pk_i) to answer the $\mathsf{QPKProcess}$ query. Before outputting, it replaces the ciphertext columns indexed by $\{i\}_{i\in\mathcal{HI}}$ in the resulting MBox and OBox by encryptions of $\mathbf{0}$. Additionally, the challenger outputs the functional keys $(\mathsf{ip.dk}^i_j)_{j,i\in[n]}$ as if they were generated without any replacements. The indistinguishability between $\mathbf{G}_2$ and $\mathbf{G}_3$ follows from the IND-CPA security of the PKE scheme, as shown in Lemma 1.

In game $\mathbf{G}_3$, the adversary's view on secrets of voters in $\mathcal{HI}$ is identical to that in the warm-up scheme, while the challenger retains the ability to program the random permutation, the one-time pads, and the $\mathsf{FH\text{-}IPFE}$ keys/ciphertexts that correspond to these voters. Let the challenge votes be $(v_i^\beta)_{i\in\mathcal{HI}}$, where $\beta \in \{0,1\}$. There exists a random permutation Π^β over n such that $\Pi^\beta(i) = i$ for $i \in [n] \setminus \mathcal{HI}$ and for any random permutation Π, applying $\Pi \circ \Pi^\beta$ to $(v_i^\beta)_{i\in[n]}$ is equivalent to applying Π to $(v_i^0)_{i\in[n]}$, assuming $v_i^\beta = v_i^0 = v_i^1$ for $i \in [n] \setminus \mathcal{HI}$.

Denoting the columns forming the matrices corresponding to $\Pi \circ \Pi^\beta$ and Π by $(\boldsymbol{p}^{\beta,i})_{i\in[n]}$ and $(\boldsymbol{p}^{0,i})_{i\in[n]}$, respectively, we have

$$\sum_{i\in\mathcal{HI}} p_j^{\beta,i} v_i^\beta = \sum_{i\in\mathcal{HI}} p_j^{0,i} v_i^0 \ \forall j \in [n]$$

by Remark 3. Using an argument similar to the one in the proof of Theorem 1, we transition from $\mathbf{G}_3$ to the following game, where there is no information on the challenge bit.

Game $\mathbf{G}_4$: In this game, instead of using $(v_i^\beta)_{i\in\mathcal{HI}}$ and $(\boldsymbol{p}^{\beta,i})_{i\in\mathcal{HI}}$, the challenger uses $(v_i^0)_{i\in\mathcal{HI}}$ and $(\boldsymbol{p}^{0,i})_{i\in\mathcal{HI}}$ to respond to QVote_β and $\mathsf{QPKProcess}$ queries, respectively. The indistinguishability between $\mathbf{G}_3$ and $\mathbf{G}_4$ is implied by the SXDH assumption in $\mathcal{PG}$ and the IND-CPA security of the $\mathsf{FH\text{-}IPFE}$ scheme.

[13] If the NIZK systems are $\mathcal{RO}$-based, we add a specific counter in $\mathcal{RO}$ queries of each system to avoid collisions across different proofs during simulation.

We finish the proof by proving the following lemma.

Lemma 1. *The transition between* $\mathbf{G}_2$ *and* $\mathbf{G}_3$ *is indistinguishable, under the IND-CPA security of the* PKE *scheme.*

Proof. Given an adversary $\mathcal{B}$ that plays against the IND-CPA security and receives the public key $\mathsf{PKE.pk}$, $\mathcal{B}$ simulates the challenger of the voting protocol as follows:

- To answer $\mathsf{QNewHonest}(\mathsf{id})$: $\mathcal{B}$ generates basis functional keys and stores the FH-IPFE secret key as in the KeyGen algorithm. To generate the ciphertext $\mathsf{ct}_{\mathsf{id}}$, it sends $(m_0 = \mathsf{ip.dk}^{\mathsf{id}}_{(1,0)}, m_1 = \mathbf{0})$ as challenge messages to the PKE encryption oracle. If m_0 is chosen, we are in game $\mathbf{G}_2$; if m_1 is chosen, we are in game $\mathbf{G}_3$. To complete the simulation, $\mathcal{B}$ simulates the proof $\pi^{\mathsf{FS}}_{\mathsf{id}}$ for $\mathsf{ct}_{\mathsf{id}}$ by programming $\mathcal{RO}$ as described in game $\mathbf{G}_2$. If any simulation fails, $\mathcal{B}$ aborts with an error called "Simulation Error". Additionally, $\mathcal{B}$ will need to run a "no-replacement" version: it generates a ciphertext $\mathsf{ct}'_{\mathsf{id}}$ that encrypts $\mathsf{ip.dk}^{\mathsf{id}}_{(1,0)}$ and stores the pair $(\mathsf{ip.dk}^{\mathsf{id}}_{(1,0)}, \mathsf{ct}'_{\mathsf{id}})$.
- To answer $\mathsf{QCor}(\mathsf{id})$: $\mathcal{B}$ returns the FH-IPFE secret key to the adversary if $\mathsf{ip.sk}_{\mathsf{id}}$ is stored and ignores the query otherwise.
- To answer $\mathsf{QPKProcess}$: Given the list $\mathsf{L} = \{\mathsf{pk}_i\}_{i\in[n]}$, $\mathcal{B}$ chooses a random permutation Π, one-time pads $\{z_j\}_{j\in[n]}$ and key randomness $\{\gamma_{\mathsf{IPE},j,i}\}_{j,i\in[n]}$. It generates $(\mathsf{pz}_i)_{i\in[n]}$ and $(\mathsf{ct}^{\mathsf{PAD}}_j)_{j\in[n]}$ and processes them as in the real game. The following modifications are applied:
  - It first generates OBox as in the $\mathsf{PKProcess}$ algorithm. For each entry in the columns indexed by $\mathcal{HI}$, $\mathcal{B}$ sends $(m_0 = z_j \cdot \mathsf{ip.dk}^i_{(0,1)} + \gamma_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)}, m_1 = \mathbf{0})$ to the PKE encryption oracle, receives a ciphertext $\mathsf{ct}^{\mathsf{OBox}}_{j,i}$, and replaces the entry $\mathsf{OBox}(j,i)$ by $\mathsf{ct}^{\mathsf{OBox}}_{j,i}$. If m_0 is encrypted, the resulting box is generated as in game $\mathbf{G}_2$ (replacement and no-replacement are equivalent in this case); if m_1 is encrypted, the resulting box is generated as in game $\mathbf{G}_3$.
  - To generate MBox, it starts with MBox_I that is patched with replaced ciphertexts ct_i where

$$\mathsf{MBox}_I(j,i) = \begin{cases} \mathsf{ct}_i & \text{if } j = i, \\ \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(0,0)}; 0) & \text{if } j \neq i. \end{cases}$$

    and then permutes and re-randomizes MBox_I by Π. On the other hand, it runs a no-replacement version by starting with MBox'_I that is instead patched with $\mathsf{ct}'_{\mathsf{id}}$, resulting in MBox'.

$$\mathsf{MBox}'_I(j,i) = \begin{cases} \mathsf{ct}_i & \text{if } j = i \text{ and } i \in \mathcal{RI}, \\ \mathsf{ct}'_i & \text{if } j = i \text{ and } i \in [n] \setminus \mathcal{RI}, \\ \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(0,0)}; 0) & \text{if } j \neq i. \end{cases}$$

    As the plaintext of ct'_i is associated with $\mathsf{ip.dk}^i_{(1,0)}$, $\mathcal{B}$ knows the plaintext $\boldsymbol{m}^{\mathsf{MBox}'}_{j,i}$ under $\mathsf{MBox}'(j,i)$ for all voters $i \in [n] \setminus \mathcal{RI}$ (those registered via $\mathsf{QNewHonest}$ queries). For each entry in the columns indexed by $\mathcal{HI}$, it sends $(m_0 = \boldsymbol{m}^{\mathsf{MBox}'}_{j,i}, m_1 = \mathbf{0})$ to the PKE encryption oracle, receives a ciphertext $\mathsf{ct}^{\mathsf{MBox}}_{j,i}$, and replaces the entry $\mathsf{MBox}(j,i)$ by $\mathsf{ct}^{\mathsf{MBox}}_{j,i}$. If m_0 is encrypted, the resulting box is generated as in game $\mathbf{G}_2$ (box-entry replacement and no-replacement are equivalent in this case); if m_1 is encrypted, the resulting box is generated as in game $\mathbf{G}_3$.
  - To simulate the decryption of $\mathsf{MBox} + \mathsf{OBox}$, $\mathcal{B}$ needs to extract $\mathsf{ip.dk}^i_{(1,0)}$ from ct_i of each voter $i \in \mathcal{RI}$ who is registered by the adversary via $\mathsf{QReqCast}$ queries. The extraction is done via the proof π^{FS}_i submitted by $\mathcal{A}$ in the form of (identity, statement, initial message, response), or $(\mathsf{aux} := \mathsf{id}_i, x, a, z)$ when using the notation in the Σ-protocol. To do that, $\mathcal{B}$ rewinds the adversary $\mathcal{A}$ up to this decryption time. If $\mathcal{A}$ queries $\mathcal{RO}$ on input (aux, x, a) to compute the challenge, then at each rewind, $\mathcal{B}$ assigns the output with a different value e'. Once a valid proof (x, a, z') is submitted by $\mathcal{A}$ in this rewinding, $\mathcal{B}$ can extract the plaintext of ct_i. This extraction fails if $\mathcal{A}$ does not need to query $\mathcal{RO}$ to compute the challenge, or if the assignment of (aux, x, a) with an output for the oracle has already been made by $\mathcal{B}$ for the zero-knowledge simulation; in both cases, $\mathcal{B}$ aborts with an error called "Extraction Error". Once all $\mathsf{ip.dk}^i_{(1,0)}$ are extracted for all $i \in \mathcal{RI}$, $\mathcal{B}$ has the basis functional keys of all registered voters in plaintext. It then performs a shuffle using Π and adds randomness $(z_j, \gamma_{\mathsf{IPE},j,i})_{j,i\in[n]}$ as if they were encrypted under PKE and processed homomorphically in the $\mathsf{PKProcess}$ algorithm.
  - $\mathsf{QReqCast}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}})$, $\mathsf{QVote}_\beta(\mathsf{id}, v_0, v_1)$, $\mathsf{QCast}(\mathsf{id}, b_{\mathsf{id}})$, $\mathsf{QFaultCrt}()$: $\mathcal{B}$ operates as the challenger does in these oracles. This is because there is no need to query the PKE encryption oracle or to perform operations involving $\mathsf{PKE.sk}$ when answering these queries.

At the end, $\mathcal{B}$ returns the output bit of the simulated ballot privacy game as its guess α for the PKE challenge bit b; on a simulation abort, it returns an independent uniform bit. Here, $b = 0$ selects $\mathbf{G}_2$ and $b = 1$ selects $\mathbf{G}_3$, independently of the voting challenge bit β, which is fixed for this transition. Let p_k be the probability that $\mathbf{G}_k$ outputs 0, for $k \in \{2, 3\}$, and let ϵ_b be the simulation-abort probability in PKE world b. Under the convention of Definition 6, the events $\mathrm{IND}_0^{\mathsf{PKE}} = 1$ and $\mathrm{IND}_1^{\mathsf{PKE}} = 0$ both correspond to $\alpha = 0$ in their respective worlds. Consequently,

$$|p_2 - p_3| \leq \mathsf{Adv}^{\mathrm{IND}^{\mathsf{PKE}}}(\lambda, \rho, \mathcal{B}) + \epsilon_0 + \epsilon_1.$$

As in [BPW12], we claim that the probability that $\mathcal{B}$ aborts in the above simulation is negligible:

- Simulation Error occurs when $\mathcal{A}$ has already queried $(\mathsf{id}_i, (\mathsf{PKE.pk}, \mathsf{ct}_i), a)$ for some voter i registered via $\mathsf{QNewHonest}$. To do this, $\mathcal{A}$ has to guess the randomness used in ct_i, which is generated honestly. The probability is negligible.
- Extraction Error: the first case occurs when $\mathcal{A}$ does not need to query $\mathcal{RO}$ to compute the challenge; it must then have guessed correctly the output of $\mathcal{RO}$ on input $(i, (\mathsf{PKE.pk}, \mathsf{ct}_i), a)$ for which it generates the proof (e.g. $i \in \mathcal{RI}$). This guess is made before $\mathcal{B}$ makes an $\mathcal{RO}$ assignment for the verification. As the output to be assigned is random, the first case occurs with negligible probability. The second case occurs when the input $(\mathsf{id}_i, (\mathsf{PKE.pk}, \mathsf{ct}_i), a)$ needs to be assigned an output for the zero-knowledge simulation. As we hardwire id_i in each proof for ct_i and simulate proofs only for i queried via the $\mathsf{QNewHon}$ oracle (e.g. $i \in [n] \setminus \mathcal{RI}$), such a case never happens. □

The ballot-privacy proof is complete because the above lemma holds. □

6 OMIX from Linearly Homomorphic Signatures

In the multi-server setting, the verifiability of an OMIX protocol is crucial for both ballot privacy and universal verifiability within the voting framework presented in Section 5. Although OMIX operates in the offline phase, the permutation-contributing process still requires sequential execution by mix-servers on encrypted basis functional keys.

However, unlike traditional mixes, OMIX shuffles a matrix of ciphertexts, requiring mix-servers to re-randomise all ciphertexts and permute matrix rows. This introduces a novel shuffling language for verification, for which no dedicated proof systems currently exist.

To address this, we enhance a tag-randomisable linear homomorphic signature (LHS) with two key properties: tag-binding unforgeability (a stronger security notion) and two-dimensional tags (a novel tag feature). The resulting scheme enables efficient verification for this new shuffling language, with verification cost independent of the number of mix-servers, rather than of the number of voters.

Using this tool, we construct an OMIX protocol whose verification time remains independent of the number of mix-servers. This construction also enables secure integration into the voting framework.

The certification authority $\mathcal{CA}$ authenticates the initial mix inputs by signing them; it is distinct from the servers that subsequently shuffle those inputs. The certified-key model assumes honest certification, so the soundness guarantee does not cover compromise of the certification signing key. Distributing certification would require a secure joint-signing protocol and a separate analysis; it is not implied solely by having one honest mix-server. Likewise, verifiability checks the permitted transformations, whereas permutation privacy additionally relies on an honest mixing contribution.

6.1 Preliminaries on Linearly Homomorphic Signatures

In this work, we rely on linearly homomorphic signatures with randomisable tags, as introduced in [HPP20], and instantiate them using the construction from [FHS19].

Definition 19 (Linearly-Homomorphic Signature (LH-Sign)). *A linearly-homomorphic signature scheme with messages in $\mathcal{M} \in \mathbb{G}^\rho$, for a cyclic group $\mathbb{G}$ of prime order p, some dimension $\rho \in \mathsf{poly}(\lambda)$, and some tag set $\mathcal{T}$, consists of eight algorithms:*

- $\mathsf{SetUp}(1^\lambda)$*: Given a security parameter λ, it outputs the global parameter* pp*, which includes the tag space $\mathcal{T}$;*
- $\mathsf{KeyGen}(\mathsf{pp}, \rho)$*: Given a public parameter* pp *and an integer ρ, it outputs a key pair* $(\mathsf{sk}, \mathsf{vk})$*, where* vk *implicitly contains* pp *and* sk *implicitly contains* vk*;*
- $\mathsf{NewTag}(\mathsf{sk})$*: Given a signing key* sk*, it outputs a tag τ and its associated secret key $\tilde{\tau}$;*

- $\mathsf{VerifTag}(\mathsf{vk}, \tau)$*: Given a verification key* vk *and a tag* τ*, it outputs* 1 *if the tag is valid and* 0 *otherwise;*
- $\mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M})$*: Given a signing key, a secret key tag* $\tilde{\tau}$ *and a vector-message* $\boldsymbol{M} = (M_i)_i \in \mathbb{G}^\rho$*, it outputs the signature* σ *under the tag* τ*;*
- $\mathsf{DerivSign}(\mathsf{vk}, \tau, (\omega_i, \boldsymbol{M}_i, \sigma_i)_{i\in[\ell]})$*: Given a public key* vk*, a tag* τ *and* ℓ *tuples of weights* $\omega_i \in \mathbb{Z}_p$ *and signed messages* $\boldsymbol{M}_i$ *with signatures* σ_i*, it outputs a signature* σ *on the vector* $\boldsymbol{M} = \sum_{i=1}^{\ell} \omega_i \cdot \boldsymbol{M_i}$ *under the tag* τ*;*
- $\mathsf{Verify}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$*: Given a verification key* vk*, a tag* τ*, a vector-message* $\boldsymbol{M}$ *and a signature* σ*, it outputs* 1 *if* $\mathsf{VerifTag}(\mathsf{vk}, \tau) = 1$ *and* σ *is also valid relative to* vk *and* τ*, and* 0 *otherwise.*
- $\mathsf{RandTag}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$*: Given a verification key* vk*, a tag* τ *and a signature* σ *on a vector-message* $\boldsymbol{M} = (M_i)_i \in \mathbb{G}^\rho$*, it outputs* τ' *and an adapted signature* σ'*.*

Correctness. For all security parameters $\lambda \in \mathbb{N}$, parameters pp generated by $\mathsf{SetUp}(1^\lambda)$, key pairs $(\mathsf{sk}, \mathsf{vk})$ generated by $\mathsf{KeyGen}(\mathsf{pp}, \rho)$, and any tags $(\tau, \tilde{\tau})$ generated by $\mathsf{NewTag}(\mathsf{sk})$, if $\sigma_i = \mathsf{Sign}(\mathsf{sk}, \tilde{\tau}, \boldsymbol{M}_i)$ are valid signatures for $i = 1, \ldots, \ell$, and if

$$\sigma = \mathsf{DerivSign}(\mathsf{vk}, \tau, \{\omega_i, \boldsymbol{M}_i, \sigma_i\}_{i=1}^{\ell})$$

for some scalars ω_i, then both

$$\mathsf{VerifTag}(\mathsf{vk}, \tau) = 1 \qquad \mathsf{Verify}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma) = 1.$$

Tag Randomisability. For any valid tuple $(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$, the tuple $(\mathsf{vk}, \tau', \boldsymbol{M}, \sigma')$ generated by $\mathsf{RandTag}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$ has a tag τ' unlinkable to τ.

Definition 20 (Unforgeability for LH-Sign). *For an LH-Sign scheme with messages in* $\mathbb{G}^\rho$*, for any adversary* $\mathcal{A}$ *that, given tags and signatures on messages* $(\boldsymbol{M}_i)_i$ *of its choice under tags* $(\tau_i)_i$ *of its choice (for Chosen-Message Attacks), outputs a valid tuple* $(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$ *with* $\tau \in \mathcal{T}$*, there must exist some tag* $\tau' \in \{\tau_i\}_i$ *and coefficients* $(\omega_i)_{i\in I_{\tau'}}$*, where* $I_{\tau'}$ *is the index set of messages already signed under* τ'*, such that*

$$\boldsymbol{M} = \sum_{i\in I_{\tau'}} \omega_i \cdot \boldsymbol{M}_i$$

with overwhelming probability.

Additional Assumptions In this work, security notions for linearly homomorphic signatures will be proved in the generic bilinear group model under the following assumptions.

Definition 21 (Extractability Assumption). *In a group* $\mathbb{G}$ *of prime order* p*, the extractability assumption states that given vectors* $(\boldsymbol{M}_j = (M_{j,i})_i)_j$ *from elements in* $\mathbb{G}$*, for any adversary* $\mathcal{A}$*, there exists an extractor* $\mathsf{Ext}^{\mathcal{A}}$ *such that whenever* $\mathcal{A}$ *produces a new vector* $\boldsymbol{M} = \sum_j \alpha_j \cdot \boldsymbol{M}_j$*,* $\mathsf{Ext}^{\mathcal{A}}$ *outputs* $(\alpha_j)_j$*.*

We will rely on the following theorem, which is proved in [HPP20], to restrict combinations of vectors.

Theorem 4 (DL-Based Restricted Combinations of Vectors). *In a group* $\mathbb{G}$ *of prime order* p*, given* n *valid Square Diffie-Hellman tuples* $(G_i, A_i = w_i \cdot G_i, B_i = w_i \cdot A_i)_{i\in[n]}$ *for random* $G_i \xleftarrow{\$} \mathbb{G}^*$ *and* $w_i \xleftarrow{\$} \mathbb{Z}_p^*$*, outputting* $(\alpha_i)_{i\in[n]}$ *such that* $(G = \sum_i \alpha_i \cdot G_i, A = \sum_i \alpha_i \cdot A_i, B = \sum_i \alpha_i \cdot B_i)$ *is a valid Square Diffie-Hellman tuple, with at least two non-zero coefficients* α_i*, is computationally hard under the* DL *assumption.*

6.2 Enhancing Linearly Homomorphic Signatures

To construct an offline mix-net, we enhance a tag-randomisable linear homomorphic signature (LHS)[14] with two key enhancements: tag-binding unforgeability (a stronger security notion) and two-dimensional tags (a novel tag feature). The resulting scheme not only enables tailored verification for this new shuffling language but also provides a building block for an OMIX protocol that is scalable in the number of mix-servers and offers simple per-ciphertext verification.

[14] In this work, we consider only linearly homomorphic signatures with randomisable tags. From this point forward, we refer to them simply as LHS.

Definition 22 (Tag-Binding Unforgeability for LH-Sign). *For a* LHS *scheme with messages in* $\mathbb{G}^\rho$*, for any adversary* $\mathcal{A}$ *that, given tags and signatures on messages* $(\boldsymbol{M}_i)_i$ *of its choice under tags* $(\tau_i)_i$ *of its choice (for Chosen-Message Attacks), outputs a set of valid tuples* $(\mathsf{vk}, \tau, \boldsymbol{M}'_j, \sigma_j)_j$ *with* $\tau \in \mathcal{T}$*, there must exist some queried tag* $\tau' \in (\tau_i)_i$ *and coefficients* $(\omega_{i,j})_{i \in I_{\tau'}}$ *corresponding to each* j*, where* $I_{\tau'}$ *is the index set of messages already signed under* τ'*, such that* $\boldsymbol{M}'_j = \sum_{i \in I_{\tau'}} \omega_{i,j} \cdot \boldsymbol{M}_i$ *with overwhelming probability.*

We show that if an LHS scheme has homomorphic verification, which is defined below, then its unforgeability implies its tag-binding unforgeability.

Definition 23 (Homomorphic Verification). *A* LHS *scheme supports homomorphic verification if, given any two signatures* $(\boldsymbol{M}_1, \sigma_1)$ *and* $(\boldsymbol{M}_2, \sigma_2)$ *that are valid (but possibly malformed) with respect to a tag* τ *and a verification key* vk*, i.e.* $\mathsf{Verify}(\mathsf{vk}, \tau, \boldsymbol{M}_i, \sigma_i) = 1$ *for* $i \in \{1, 2\}$*, for any scalars* $\alpha_1, \alpha_2 \in \mathbb{Z}_p$*, the signature* σ *derived from* $\alpha_1 \cdot \sigma_1 + \alpha_2 \cdot \sigma_2$ *is also valid with respect to the message* $(\alpha_1 \cdot \boldsymbol{M}_1 + \alpha_2 \cdot \boldsymbol{M}_2)$ *and the tuple* (τ, vk)*.*

We have the following main theorem.

Theorem 5 (Tag-Binding Unforgeability from Homomorphic Verification). *Given an* LHS *scheme, if it supports homomorphic verification, it is also tag-binding unforgeable (as defined in Definition 22).*

Proof. We consider an adversary $\mathcal{A}$ in the tag-binding unforgeability game of the scheme. We then construct an adversary $\mathcal{B}$ that runs $\mathcal{A}$ to break the unforgeability.

Upon receiving the NewTag and Sign queries from $\mathcal{A}$, the adversary $\mathcal{B}$ simply forwards these queries to the oracles of its game to simulate answers. We then denote by $(\boldsymbol{M}_i)_i$ the set of queried messages under tags $(\tau_i)_i$ (possibly equal across i) of which both are chosen by $\mathcal{A}$. We also denote by V_{τ_i} the sub-vector space generated by messages queried to Sign under τ_i.

In the end when $\mathcal{A}$ outputs the set of forgeries $(\mathsf{vk}, \tau, \boldsymbol{M}'_j, \sigma_j)_{j \in [\ell]}$, the adversary $\mathcal{B}$ then generates randomly $(\alpha_j)_{j \in [\ell]} \xleftarrow{\$} \mathbb{Z}_p^\ell$ and outputs its forgery as

$$(\mathsf{vk}, \tau, \boldsymbol{M}^\star = \sum_{j \in [\ell]} \alpha_j \cdot \boldsymbol{M}'_j, \sigma^\star = \sum_{j \in [\ell]} \alpha_j \cdot \sigma_j)_{j \in [\ell]}.$$

By the homomorphic verification, if $\mathcal{A}$'s forgeries are valid, then $\mathcal{B}$'s forgery is also valid. It then suffices to prove that if there does not exist a queried tag $\tau' \in (\tau_i)_i$ such that $\boldsymbol{M}'_j \in V_{\tau'}$ for all $j \in [\ell]$, then with overwhelming probability, there does not exist a queried tag $\tau' \in (\tau_i)_i$ such that $\boldsymbol{M}^\star \in V_{\tau'}$. We assume that the former case already happened.

By the assumption, for each $\tau' \in (\tau_i)_i$, we have a non-empty index subset $\bar{\mathcal{I}}'_\tau \subset [\ell]$ such that $\boldsymbol{M}'_k \notin V_{\tau'}$ if $k \in \bar{\mathcal{I}}'_\tau$. Let $U_{\tau'}$ represent the vector space generated by $V_{\tau'}$ and vectors $(\boldsymbol{M}'_k)_{k \in \bar{\mathcal{I}}_{\tau'}}$. By construction, $U_{\tau'}$ strictly contains $V_{\tau'}$. WLOG, we assume that $\{\boldsymbol{b}_t\}_{t \in N}$ is a subset of a basis of $U_{\tau'}$ but does not lie in $V_{\tau'}$, for some N subject to $1 \leq N \leq |\mathcal{I}'_\tau|$. By the assumption, the coefficient tuple $\boldsymbol{c}_k$ of each message in $(\boldsymbol{M}'_k)_{k \in [\ell], k \notin \bar{\mathcal{I}}'_\tau}$ with respect to basis vectors $\{\boldsymbol{b}_t\}_{t \in N}$ is $(0, .., 0)$ (trivial) as these messages lie in $V_{\tau'}$, while that of each message in $(\boldsymbol{M}'_k)_{k \in \bar{\mathcal{I}}'_\tau}$ is non-trivial.

Therefore, the forged message $\boldsymbol{M}^\star$ lies in $V_{\tau'}$ if and only if the equality $\sum_{k \in \bar{\mathcal{I}}'_\tau} \alpha_k \boldsymbol{c}_k = \boldsymbol{0}$ holds. Since each $\boldsymbol{c}_k$ for $k \in \bar{\mathcal{I}}'_\tau$ is non-trivial, the number of solutions for $(\alpha_k)_{k \in \bar{\mathcal{I}}'_\tau}$ is upper-bounded by $p^{|\mathcal{I}'_\tau| - 1}$. Therefore, the probability that a random $(\alpha_k)_{k \in \bar{\mathcal{I}}'_\tau}$ becomes a solution is upper-bounded by $1/p$. As $p = \mathcal{O}(2^\lambda)$, the probability that $\boldsymbol{M}^\star$ lies in $V_{\tau'}$ is negligible.

As the number of queried tags $(\tau_i)_i$ is polynomial in λ, then the probability that $\boldsymbol{M}^\star$ lies in V_{τ_i} for some i is also negligible. This completes the proof. □

Remark 4. The LHS scheme, originally constructed for signing projective equivalence classes in [FHS19] and subsequently shown to support the signing of linear spaces in [HPP20], is tag-binding unforgeable.

We recall the verification algorithm of the scheme as follows:

- the parameter is $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, \mathcal{T} = \mathbb{G}_1 \times \mathbb{G}_2, \rho)$;
- the verification key is $\mathsf{vk} = ([s_i]_2)_{i \in [\rho]}$ where $(s_i)_{i \in [\rho]}$ is the signing key;
- a vector-message is $\boldsymbol{M} = (M_i)_{i \in [\rho]} \in \mathbb{G}_1^\rho$;
- a tag is $\tau = (\tau_1 \in \mathbb{G}_1, \tau_2 \in \mathbb{G}_2)$;

– a signature σ is in $\mathbb{G}_1$.

The verification then takes as input a tuple $(\mathsf{pp}, \mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$ and checks whether the equalities $e(\sigma, \tau_2) = \sum_{i\in[\rho]} e(M_i, [s_i]_2)$ and $e(\tau_1, [1]_2) = e([1]_1, \tau_2)$ hold. Therefore, given two tuples $(\mathsf{vk}, \tau, \boldsymbol{M}_1, \sigma_1)$ and $(\mathsf{vk}, \tau, \boldsymbol{M}_2, \sigma_2)$ that are verified as valid, one can forge a signature under (vk, τ) for any message $\boldsymbol{M} = \alpha_1 \cdot \boldsymbol{M}_1 + \alpha_2 \cdot \boldsymbol{M}_2$ by computing $\sigma = \alpha_1 \cdot \sigma_1 + \alpha_2 \cdot \sigma_2$. The tuple $(\mathsf{vk}, \tau, \boldsymbol{M})$ is valid by the following equalities:

$$\begin{aligned} e(\sigma, \tau_2) &= e(\alpha_1 \cdot \sigma_1 + \alpha_2 \cdot \sigma_2, \tau_2) \\ &= \alpha_1 \cdot e(\sigma_1, \tau_2) + \alpha_2 \cdot e(\sigma_2, \tau_2) \\ &= \alpha_1 \cdot \sum_{i\in[\rho]} e(M_{1,i}, [s_i]_2) + \alpha_2 \cdot \sum_{i\in[\rho]} e(M_{2,i}, [s_i]_2) \\ &= \sum_{i\in[\rho]} e(\alpha_1 \cdot M_{1,i} + \alpha_2 \cdot M_{2,i}, [s_i]_2) \end{aligned}$$

Two-Dimensional Tags We want to ensure that elements can be combined (e.g. for re-randomisation) only if they correspond to the same position in the matrix. Running two LHS instances, one for rows and one for columns, fails to meet this security requirement. We therefore introduce LHS with two-dimensional tags, where one dimension corresponds to the row and the other to the column. As for one-dimensional tags, the security guarantees that only messages signed under tags of the same row and column can be combined (e.g. equal tags). At the same time, public randomisation of row (sub-)tags is available, and the tag-binding property (see Definition 26) still holds with respect to these row tags.

Definition 24 (Two-Dimensional Tags). *A* LHS *scheme (as defined in Definition 19) admits two-dimensional tags if the tag set $\mathcal{T}$ can be split into $\mathcal{T}_1 \times \mathcal{T}_2$ where $\mathcal{T}_1$ is a randomisable set. Specifically, the scheme admits a* RandTag *algorithm as follows:* $\mathsf{RandTag}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$ *takes as input a verification key* vk*, a tag $\tau = (\tau_1, \tau_2)$ where $\tau_1 \in \mathcal{T}_1$ and $\tau_2 \in \mathcal{T}_2$, and a signature σ on a vector-message $\boldsymbol{M} = (M_i)_i \in \mathbb{G}^\rho$; it outputs $\tau' = (\tau'_1, \tau_2)$ where τ'_1 is a re-randomisation of τ_1 and an adapted signature σ'.*

The following unlinkability definition for tags in $\mathcal{T}_1$ ensures that a re-randomized row tag reveals no information about its original tag, allowing an unlinkable shuffle of the rows.

Definition 25 ($\mathcal{T}_1$-Tag Unlinkability). *For any valid tuple $(\mathsf{vk}, \tau = (\tau_1, \tau_2), \boldsymbol{M}, \sigma)$, the tuple $(\mathsf{vk}, \tau' = (\tau'_1, \tau_2), \boldsymbol{M}, \sigma')$ generated by $\mathsf{RandTag}(\mathsf{vk}, \tau, \boldsymbol{M}, \sigma)$ has a tag τ'_1 unlinkable to τ_1.*

From now on, we will refer to LHS schemes with two-dimensional tags as 2-Tag-LHS, to distinguish them from one-dimensional-tag LHS schemes.

Definition 26 (Tag-Binding Unforgeability for 2-Tag-LHS). *Given a 2-tag-*LHS *scheme, for any adversary $\mathcal{A}$ that, given tags and signatures on messages $(\boldsymbol{M}_i)_i$ of its choice under two-dimensional tags $(\tau_{1,i}, \tau_{2,i})_i$ of its choice (for Chosen-Message Attacks), outputs a set of valid tuples $(\mathsf{vk}, \tau_j = (\tau_1, \tau_{2,j}), \boldsymbol{M}'_j, \sigma_j)_j$ with $\tau_1 \in \mathcal{T}_1$ and each $\tau_{2,j} \in (\tau_{2,i})_i$, there must exist some tag τ'_1 such that each $(\tau'_1, \tau_{2,j}) \in (\tau_{1,i}, \tau_{2,i})_i$, and coefficients $(\omega_{i,j})_{i\in I_{\tau'_1,\tau_{2,j}},j}$, where each $I_{\tau'_1,\tau_{2,j}}$ is the set of messages already signed under $(\tau'_1, \tau_{2,j})$, such that*

$$\boldsymbol{M}'_j = \sum_{i\in I_{\tau'_1,\tau_{2,j}}} \omega_{i,j} \cdot \boldsymbol{M}_i$$

with overwhelming probability.

6.3 Construction of 2-Tag-LHS

We apply the generic technique of imposing a tag on the one-time LHS scheme in [HPP20] to impose a tag on any existing LHS with a randomisable tag set, denoted as $\mathsf{LHS}_1 = (\mathsf{SetUp}_1, \mathsf{KeyGen}_1, \mathsf{NewTag}_1, \mathsf{VerifTag}_1, \mathsf{Sign}_1, \mathsf{DerivSign}_1, \mathsf{Verify}_1, \mathsf{RandTag}_1)$, as in the following.

Theorem 6 (Tag-Binding Unforgeability for 2-tag-LHS). *If LHS_1 is a tag-binding unforgeable LHS, then 2-tag-*LHS*, as constructed in Figure 12, is also tag-binding unforgeable under the DL assumption in the generic bilinear group model.*

$\underline{\mathsf{SetUp}(1^\lambda)}$: It runs $\mathsf{SetUp}_1(1^\lambda)$ to obtain pp_1, which includes the tag space $\mathcal{T}_1$ and adds the tag space $\mathcal{T}_2 = (\mathbb{Z}_p^* \times \mathbb{G}_1^*)$. It outputs $\mathsf{pp} = (\mathsf{pp}_1, \mathcal{T}_2)$.

$\underline{\mathsf{KeyGen}(\mathsf{pp}, \rho)}$: It runs $(\mathsf{sk}_1, \mathsf{vk}_1) \leftarrow \mathsf{KeyGen}_1(\mathsf{pp}_1, \rho + 3)$ and outputs $\mathsf{sk} = \mathsf{sk}_1$ and $\mathsf{vk} = \mathsf{vk}_1$.

$\underline{\mathsf{NewTag}_{\mathcal{T}_1}(\mathsf{sk})}$: It outputs a tag in $\mathcal{T}_1$ as $(\tau_1, \tilde{\tau}_1) \leftarrow \mathsf{NewTag}_1(\mathsf{sk}_1)$.

$\underline{\mathsf{NewTag}_{\mathcal{T}_2}(\mathsf{sk})}$: It chooses a scalar $w \xleftarrow{\$} \mathbb{Z}_p^*$ and an element $H \xleftarrow{\$} \mathbb{G}_1^*$, and outputs $\tilde{\tau}_2 = \tau_2 = (w, H)$.

$\underline{\mathsf{VerifTag}_{\mathcal{T}_1}(\mathsf{vk}, \tau_1)}$: It checks whether $\mathsf{VerifTag}(\mathsf{vk}, \tau_1) = 1$ or not.

$\underline{\mathsf{VerifTag}_{\mathcal{T}_2}(\mathsf{vk}, \tau_2)}$: It checks whether $\tau_2 = (w, H) \in \mathbb{Z}_p^* \times \mathbb{G}_1^*$ or not.

$\underline{\mathsf{Sign}(\mathsf{sk}, \tilde{\tau}_1, \tilde{\tau}_2 = (w, H), \boldsymbol{M})}$: It uses $\tilde{\tau}_2$ to extend $\boldsymbol{M} \in \mathbb{G}_1^\rho$ into $\boldsymbol{M}' = (\boldsymbol{M}, H, w \cdot H, w^2 \cdot H) \in \mathbb{G}_1^{\rho+3}$, and outputs $\sigma \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \tilde{\tau}_1, \boldsymbol{M}')$.

$\underline{\mathsf{DerivSign}(\mathsf{vk}, \tau_1, \tau_2 = (w, H), \{\omega_i, \boldsymbol{M}_i, \sigma_i\}_{i \in [\ell]})}$: It extends each $\boldsymbol{M}_i$ into the signed message $\boldsymbol{M}'_i = (\boldsymbol{M}_i, H, w \cdot H, w^2 \cdot H)$ in σ_i and computes $\sigma \leftarrow \mathsf{DerivSign}_1(\mathsf{vk}_1, \tau_1, \{\omega_i, \boldsymbol{M}'_i, \sigma_i\}_{i \in [\ell]})$ and $\tau'_2 = (w, H' = \sum_i \omega_i \cdot H)$. It outputs $(\sigma, \tau_1, \tau'_2)$.

$\underline{\mathsf{Verify}(\mathsf{vk}, \tau_1, \tau_2 = (w, H), \boldsymbol{M}, \sigma)}$: It extends $\boldsymbol{M}$ into $\boldsymbol{M}' = (\boldsymbol{M}, H, w \cdot H, w^2 \cdot H)$ and checks whether $\mathsf{VerifTag}_{\mathcal{T}_1}(\mathsf{vk}, \tau_1) = 1$ and $\mathsf{VerifTag}_{\mathcal{T}_2}(\mathsf{vk}, \tau_2) = 1$ and $\mathsf{Verify}_1(\mathsf{vk}_1, \tau_1, \boldsymbol{M}', \sigma) = 1$ or not.

$\underline{\mathsf{RandTag}(\mathsf{vk}, \tau_1, \tau_2 = (w, H), \boldsymbol{M}, \sigma)}$: It extends $\boldsymbol{M}$ into $\boldsymbol{M}' = (\boldsymbol{M}, H, w \cdot H, w^2 \cdot H)$ and randomises τ_1 by running $(\tau'_1, \sigma') \leftarrow \mathsf{RandTag}_1(\mathsf{vk}_1, \tau_1, \boldsymbol{M}', \sigma)$. It outputs (τ'_1, τ_2) and σ'.

Fig. 12: Construction of LHS with two-dimensional tags.

Proof. We consider an adversary $\mathcal{A}$ in the tag-binding unforgeability game of 2-tag-LHS. We then construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to either break the tag-binding unforgeability of LHS_1 or the DL assumption in the generic bilinear group model as follows:

- since the $\mathcal{T}_2$-tags are fully public, any $\mathsf{NewTag}_{\mathcal{T}_2}$-query is answered by a random pair (w, H);
- any $\mathsf{NewTag}_{\mathcal{T}_1}$-query is answered by forwarding a query to NewTag_1 of the LHS_1 security game;
- any Sign-query on $(\tau_1, \tau_2 = (w, H), \boldsymbol{M})$ is answered by first extending $\boldsymbol{M}$ into $\boldsymbol{M}' = (\boldsymbol{M}, H, w \cdot H, w^2 \cdot H)$ and forwarding $(\tau_1, \boldsymbol{M}')$ to the Sign_1-query of the LHS_1 security game.

We then denote by $(\boldsymbol{M}_i)_i$ the set of queried messages under two-dimensional tags $(\tau_{1,i}, \tau_{2,i} = (w_i, H_i))_i$, all of which are chosen by $\mathcal{A}$; tags in either the first or second component may be equal across i when the same tag is used several times. Furthermore, $(\boldsymbol{M}'_i)_i$ are the extended messages of $(\boldsymbol{M}_i)_i$ with their corresponding tags.

Upon receiving the valid 2-tag-LHS forgeries $(\mathsf{vk}, \tau_j = (\tau_1, \tau_{2,j}), \boldsymbol{M}^\star_j, \sigma_j)_j$ from $\mathcal{A}$ where $\tau_1 \in \mathcal{T}_1$ and each $\tau_{2,j}$ is already queried, $\mathcal{B}$ extends $\boldsymbol{M}^\star_j$ into $\boldsymbol{M}^{\star\prime}_j$ by using $\tau_{2,j} = (w^\star_j, H^\star_j)$ and outputs valid LHS_1 forgeries $(\mathsf{vk}, \tau_1, \boldsymbol{M}^{\star\prime}_j, \sigma_j)_j$. Unless the tag-binding unforgeability of LHS_1 is broken, there must exist a tag τ'_1 queried to Sign_1 and coefficients $(\omega_{i,j})_{i \in I_{\tau'_1}, j}$ where $I_{\tau'_1}$ is the index set of extended messages already signed under τ'_1, such that

$$\boldsymbol{M}^{\star\prime}_j = \sum_{i \in I_{\tau'_1}} \omega_{i,j} \cdot \boldsymbol{M}'_i \tag{2}$$

with overwhelming probability. It thus suffices to restrict the index set in the last equality from $I_{\tau'_1}$ to $I_{\tau'_1, \tau_{2,j}}$, where the latter is the index set of messages already signed under $(\tau'_1, \tau_{2,j})$.

The last three components of the LHS vector and the RHS vector in Equality (2) give

$$(H^\star_j, w^\star_j \cdot H^\star_j, w^{\star 2}_j \cdot H^\star_j) = \left(\sum_{i \in I_{\tau'_1}} \omega_{i,j} \cdot H_i, \sum_{i \in I_{\tau'_1}} \omega_{i,j} \cdot (w_i \cdot H_i), \sum_{i \in I_{\tau'_1}} \omega_{i,j} \cdot (w_i^2 \cdot H_i) \right).$$

This means that the LHS_1-forgery on j (the RHS triple) forms a square Diffie-Hellman triple. We assume that the $\mathcal{T}_2$-tags queried with τ_1 are contained in the following set of *distinct* items $(\tau_{2,k} = (w_k, H_k))_k$. By

combining identical $\mathcal{T}_2$-tags on the RHS and summing in each β_k the coefficients $(\omega_{i,j})_i$ that correspond to the same triple $(H_k, w_k \cdot H_k, w_k^2 \cdot H_k)$, we obtain the RHS

$$\left(\sum_k \beta_k \cdot H_k, \sum_k \beta_k \cdot (w_k \cdot H_k), \sum_k \beta_k \cdot (w_k^2 \cdot H_k)\right).$$

From Theorem 4, unless the DL assumption is broken, at most one coefficient is non-zero: either none or β_K, and so at most one $\mathcal{T}_2$-tag is represented: none or (w_K, H_K). Hence $\boldsymbol{M}_j^\star$ is either $(1, \ldots, 1)$ or $\sum_{i \in I_{\tau_1', \tau_{2,K}}} \omega_{i,j} \cdot \boldsymbol{M}_i$. If the latter case occurs, then one has

$$(H_j^\star, w_j^\star \cdot H_j^\star, {w_j^\star}^2 \cdot H_j^\star) = \big(\beta_K \cdot H_K, \beta_K \cdot (w_K \cdot H_K), \beta_K \cdot (w_K^2 \cdot H_K)\big)$$

which implies $w_K = w_j^\star$. This concludes $\boldsymbol{M}_j^\star = \sum_{i \in I_{\tau_1', \tau_{2,j}}} \omega_{i,j} \cdot \boldsymbol{M}_i$. □

6.4 OMIX Construction

Based on the computations that an OMIX protocol should perform, as described in Section 5, we present an OMIX construction with three phases: Setup (see Figure 13), Shuffle (see Figure 14), and Verification (see Figure 15).

We outline the intuition behind these phases below and provide the proofs in Section 7.2.

Setup The setup outputs a pairing group $\mathcal{PG}$[15] as parameters for the two-dimensional-tag LHS (2-tag-LHS) scheme. OMIX adopts the PKE scheme, which is instantiated in $\mathbb{G}_1$ of $\mathcal{PG}$ from the voting protocol, as a public parameter. Additionally, the setup provides a $\mathcal{RO}$-modeled hash function H mapping onto $\mathbb{G}_1$ and a common reference string for the $\mathsf{NIZK}_{\mathsf{DH}}$ proof system to verify Diffie-Hellman (DH) tuples.

Key Generation As in the voting protocol's local secret-key generation phase, each voter generates an FH-IPFE secret key and submits encrypted basis functional keys for registration. We designate the vote-slot decrypting key $(1, 0)$, encrypted under PKE, as the voter's secret key. Additionally, each mix-server registers for participation in OMIX by submitting its verification key for a standard signature scheme.

The OMIX protocol requires a Certificate Authority ($\mathcal{CA}$) that holds the secret signing key for the 2-tag-LHS scheme. The role of $\mathcal{CA}$ is to certify (by signing) mix inputs contributed by voters during registration. The soundness of OMIX verification relies on the secrecy of this signing key, which is considered the master secret key of the protocol. Consequently, the corresponding 2-tag-LHS verification key serves as the master public key, later used for mix verification.

Certification Once voter and mix-server registration is complete, $\mathcal{CA}$ certifies the mix-servers' verification keys, for example by publishing a list. Certifying mix inputs from voters consists of two steps.

The first step is to arrange the basis functional keys according to an identity matrix (initial permutation). The keys $(1, 0)$ of the voters are placed along the diagonal. Each column i is filled with the key $(0, 0)$ of voter i. To match the encrypted key $(1, 0)$, each key $(0, 0)$ is parsed as encrypted under PKE with null randomness. Additionally, a PKE-randomising ciphertext (or PKE randomiser) is set. This step is public and deterministic.

In the second step, $\mathcal{CA}$ signs each encrypted key in the matrix using the 2-tag-LHS secret key, assigning a two-dimensional tag corresponding to the key's row and column indices. To enable verifiable re-randomisation of the PKE layer of these keys, $\mathcal{CA}$ also signs the PKE randomiser under tags corresponding to each matrix position. One has a matrix of 2-tag-LHS signatures for the PKE randomiser.

An important remark is that soundness cannot be obtained with 2-tag-LHS alone, as the adversary can mount a remove-and-replace attack: it removes one row, duplicates another row to fill in, re-randomises the row tags and shuffles. Due to row-tag unlinkability, such an attack would pass verification, as all signatures are valid. To counter this, we key the rows by introducing an additional column of their hash outputs, which $\mathcal{CA}$ also signs. Mix-servers re-randomise these row keys by multiplying them by a secret exponent. By the Diffie–Hellman assumption, unlinkability is preserved, while a DH-tuple proof ensures that the sum of the initial keys and that of the mixed keys are equal up to this exponent. This prevents remove-and-replace attacks unless the adversary can compute the discrete-logarithm relation between the hash outputs.

The mix box is initialised with the identity matrix of functional keys and their signatures; the matrix of PKE-randomiser signatures; and the column of row-key signatures. All signed elements are tagged by their respective row and column indices.

[15] This is the same group in which the PKE and FH-IPFE schemes are instantiated.

Mix On input a mix box, each server re-randomises the functional-key ciphertexts and the row keys, and adapts their signatures correspondingly. The server re-randomises the row tags and shuffles the row indices of all signed elements and their signatures. It generates a proof for the DH-tuple formed by the input row keys and the shuffled row keys. The last step is to authenticate the mixed output by signing the proof using a standard signature scheme. The mix-server passes the shuffled box to the next server, along with the proof and its signature.

Verification This phase is described in Figure 15, where any party can verify that the output mix box is contributed by all registered servers (via standard signature verification), that the initial mix box is correctly signed (via 2-tag-LHS verification), and that the output mix box is correctly permuted and re-randomised from the initial mix box (via 2-tag-LHS and DH-proof verification).

We prove the soundness of the protocol in Theorem 7 and unlinkability in Theorem 8. A stronger form of unlinkability, which enables permutation extractability during the simulation of the unlinkability proof, ensures that the protocol can be securely integrated into the voting system. Additionally, as the voting protocol requires OPAD, we provide a construction in Figure 17 and the corresponding security proofs in Section 7.3.

We also note that, by relying on 2-tag-LHS, the OMIX protocol requires trust in the certificate authority, whose role is confined to certifying the initial mix inputs before mixing begins. One possible extension is to distribute this initial certification among multiple authorities, which could then go offline. Such an extension would require a secure joint-signing protocol and a separate analysis; it is not part of the present construction.

To keep proof size and online verification cost independent of the number of mix-servers, we adopt the strategy proposed in [HPP20]. This approach uses malleable NIZK systems, such as Groth-Sahai proofs [GS08], allowing servers to update a global proof for the re-randomised DH-tuple in each round. At the end of the mixing process, all servers verify and sign this global proof using the multi-signature scheme of Boneh-Drijvers-Neven [BDN18].

7 Security of Offline Protocols and Their Integration

7.1 Auxiliary Security Notions for Offline Protocols

For the security of OMIX and OPAD in the multi-server setting, we introduce two notions—soundness and unlinkability—adapted from [HPP20] to the context of mixing eligible voters' public keys. Intuitively, soundness ensures that any protocol execution accepted as correct produces an output matrix of ciphertexts equivalent to the input matrix (e.g., up to an authorized transformation). Unlinkability, in turn, ensures that an observer cannot link the output matrix to the transformation applied to the input matrix.

Definition 27 (Soundness). *An offline protocol $\mathcal{P}_{\mathsf{OFF}}$ between multiple servers for a family of authorized transformation sets $\{\mathcal{T}_{\lambda,n}\}_{\lambda,n}$ is said to be sound in the certified key setting if any PPT adversary $\mathcal{A}$ has a negligible success probability in the following experiment:*

1. *On input a security parameter λ and a number of users n, the challenger generates public parameters of $\mathcal{P}_{\mathsf{OFF}}$ as pp and master keys as $(\mathsf{msk}, \mathsf{mpk})$. It provides $(\mathsf{pp}, \mathsf{mpk})$ to $\mathcal{A}$.*
2. *Upon receiving $(\mathsf{pp}, \mathsf{mpk})$, the adversary $\mathcal{A}$ does as follows:*
   - *It determines the set of corrupted users $\mathcal{CS} \subset [n]$ and generates their public keys $(\mathsf{pk}_i)_{i\in\mathcal{CS}}$ itself.*
   - *For $i \in \mathcal{CS}$, it proves knowledge of the secret keys sk_i for pk_i to obtain certifications $\mathcal{CA}.\Sigma_i$, which the challenger generates using msk.*
3. *The challenger generates the keys of the honest users $(\mathsf{sk}_i, \mathsf{pk}_i)_{i\in[n]\setminus\mathcal{CS}}$ and certifies their public keys by generating $(\mathcal{CA}.\Sigma_i)_{i\in[n]\setminus\mathcal{CS}}$.*
4. *The initial mix box is thus defined by $\mathsf{MBox} = (\mathcal{CA}.\Sigma_i)_{i\in[n]}$.*
5. *The adversary mixes MBox in a provable way, resulting in $(\mathsf{MBox}', \mathsf{proof})$.*

The adversary wins if $\mathsf{MixVerif}\left(\mathsf{MBox}, \mathsf{MBox}', \mathsf{proof}\right) = 1$, *but there does not exist an authorized transformation $T \in \mathcal{T}_{\lambda,n}$ such that*

$$\mathsf{MBox}' = T(\mathsf{MBox})$$

Under this definition, our offline mix-net protocol assumes that all user inputs are encapsulated in their public keys and certified by the authority before being added to the mix box. In the voting context, the inputs are functional keys, which are mixed and eventually decrypted into functional keys for the

$\underline{\mathsf{OMIX.Setup}(1^\lambda)}$:
$\mathsf{LHS.pp} := \mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e) \leftarrow$ 2-tag-$\mathsf{LHS.SetUp}(1^\lambda)$;
$\mathsf{NIZK_{DH}.pp} \leftarrow \mathsf{NIZK_{DH}.SetUp}(1^\lambda)$; $(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.SetUp}(1^\lambda, \mathbb{G}_1)$;
$H : \{0,1\}^* \to \mathbb{G}_1$;
Let $\mathsf{Mix.pp} = (\mathcal{PG}, \mathsf{LHS.pp}, \mathsf{NIZK_{DH}.pp}, \mathsf{PKE.pk}, H)$.

$\underline{\mathsf{OMIX.KeyGen}(\mathsf{Mix.pp})}$:
Certificate Authority $\mathcal{CA}$ generates $(\mathsf{LHS.sk}, \mathsf{LHS.vk}) \leftarrow$ 2-tag-$\mathsf{LHS.KeyGen}()$ and sets $\mathsf{msk} = \mathsf{LHS.sk}$ and $\mathsf{mpk} = \mathsf{LHS.vk}$.
Each voter i for $i \in [n]$ generates FH-IPFE keys

$$\mathsf{sk}_i := \mathsf{ip.dk}^i_{(1,0)};\ \mathsf{pk}_i := (\mathsf{id}_i, \mathsf{ct}_i, \mathsf{ip.dk}^i_{(0,0)})$$

where $\mathsf{ct}_i = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(1,0)})$.
Each mix-server $\mathcal{S}^{\mathsf{mix}}_s$ for $s \in [N_{\mathsf{mix}}]$ generates Sig keys $(\mathsf{SK}^{\mathsf{mix}}_s, \mathsf{VK}^{\mathsf{mix}}_s) \leftarrow \mathsf{Sig.SetUp}(1^\lambda)$.

$\underline{\mathsf{OMIX.Certify}(\mathsf{msk}, (\mathsf{pk}_i)_{i\in[n]}, (\mathsf{VK}^{\mathsf{mix}}_s)_{s\in[N_{\mathsf{mix}}]})}$[a]:
Starting from an identity matrix $\boldsymbol{I}_{n\times n}$, each column vector $\boldsymbol{e}_i$ corresponds to each voter i, initially with 1 at positions (i,i), and 0 in all other entries of the column.
Publicly Arrange: for each $i \in [n]$,

1. $\mathcal{CA}$ parses ${}^bC_{i,i} = \mathsf{ct}_i$;
2. for each row $j \neq i$, $\mathcal{CA}$ forms $C_{j,i} \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathsf{ip.dk}^i_{(0,0)}; 0)$;
3. $\mathcal{CA}$ forms a PKE randomiser $C_{\mathsf{PKE}} \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathbf{0}; 1)$;

Sign with msk

1. $\mathcal{CA}$ generates one row tag for each $j \in [n]$ and one column tag for each $i \in [n+1]$:

$$(\tau_j, \tilde{\tau}_j) \leftarrow \text{2-tag-}\mathsf{LHS.NewTag}_{\mathcal{T}_1}(\mathsf{LHS.sk}), \quad j \in [n];$$
$$(\theta_i, \tilde{\theta}_i) \leftarrow \text{2-tag-}\mathsf{LHS.NewTag}_{\mathcal{T}_2}(\mathsf{LHS.sk}), \quad i \in [n+1].$$

2. For each $(j,i) \in [n] \times [n]$, $\mathcal{CA}$ signs $\bar{C}_{j,i} = ([\![1]\!]_1, C_{j,i})$ and $\bar{C}_{\mathsf{PKE}} = ([\![0]\!]_1, C_{\mathsf{PKE}})$ under the two-dimensional tag (τ_j, θ_i):

$$\sigma_{j,i} \leftarrow \text{2-tag-}\mathsf{LHS.Sign}(\mathsf{LHS.sk}, \tilde{\tau}_j, \tilde{\theta}_i, \bar{C}_{j,i});$$
$$\Sigma_{\mathsf{PKE},j,i} \leftarrow \text{2-tag-}\mathsf{LHS.Sign}(\mathsf{LHS.sk}, \tilde{\tau}_j, \tilde{\theta}_i, \bar{C}_{\mathsf{PKE}}).$$

3. $\mathcal{CA}$ signs each $\boldsymbol{G}_j := (G_0 = H(0), G_{1,j} = H(j))$ under the tag (τ_j, θ_{n+1}):

$$\Sigma_{\boldsymbol{G}_j} \leftarrow \text{2-tag-}\mathsf{LHS.Sign}(\mathsf{LHS.sk}, \tilde{\tau}_j, \tilde{\theta}_{n+1}, \boldsymbol{G}_j)$$

The certification to a voter i is $\mathcal{CA}.\Sigma_i = (C_{j,i}, \sigma_{j,i}, \Sigma_{\mathsf{PKE},j,i})_{j\in[n]}, \theta_i$.
The initial box is $\mathsf{MBox}^{(0)} = \{(\mathcal{CA}.\Sigma_i)_{i\in[n]}, (\tau_j, G_{1,j}, \Sigma_{\boldsymbol{G}_j})_{j\in[n]}, \theta_{n+1}, G_0, C_{\mathsf{PKE}}\}$.

[a] We assume that all servers' verification keys $\mathsf{Sig.vk}_s$ are implicitly certified.
[b] In this protocol, we use the uppercase letter C to denote ElGamal-ciphertext vectors.

Fig. 13: Setup, key generation and certification of OMIX.

$\underline{\mathsf{OMIX.Shuffle}(\mathsf{SK}_s^{\mathsf{mix}}, \mathsf{MBox}^{(s-1)}, (\mathsf{proof}^{(k)}, \mathsf{sig}_{\mathsf{mix}}^{(k)})_{k\in[s-1]}):}$

Each server $\mathcal{S}_s$ takes input $\mathsf{MBox}^{(s-1)}$ as

$$\{(C_{j,i}, \sigma_{j,i}, \Sigma_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau_j, G_{1,j}, \Sigma_{\boldsymbol{G}_j})_{j\in[n]}, (\theta_i)_{i\in[n+1]}, G_0, C_{\mathsf{PKE}}\}$$

1. It generates random $\gamma_{\mathsf{PKE},j,i}$ and randomises $C_{j,i}$ for each (j,i):
$$C'_{j,i} = C_{j,i} + \gamma_{\mathsf{PKE},j,i} \cdot C_{\mathsf{PKE}}$$
2. It accordingly adapts each signature $\sigma_{j,i}$ to obtain $\sigma'_{j,i}$ for $\bar{C}'_{j,i} = (\llbracket 1 \rrbracket_1, C'_{j,i})$
$$\sigma'_{j,i} \leftarrow \text{2-tag-}\mathsf{LHS.DerivSign}(\mathsf{LHS.vk}, (\tau_j, \theta_i), (\gamma_{\mathsf{PKE},j,i}, \bar{C}_{\mathsf{PKE}}, \Sigma_{\mathsf{PKE},j,i}), (1, \bar{C}_{j,i}, \sigma_{j,i})).$$
3. It generates random β and randomises G_0 and $G_{1,j}$ for each $j \in [n]$:
$$G'_0 = \beta \cdot G_0; \qquad G'_{1,j} = \beta \cdot G_{1,j}$$
4. It adapts each signature $\Sigma_{\boldsymbol{G}_j}$ into $\Sigma'_{\boldsymbol{G}_j}$ for $\boldsymbol{G}'_j = (G'_0, G'_{1,j})$:
$$\Sigma'_{\boldsymbol{G}_j} \leftarrow \text{2-tag-}\mathsf{LHS.DerivSign}(\mathsf{LHS.vk}, (\tau_j, \theta_{n+1}), (\beta, \boldsymbol{G}_j, \Sigma_{\boldsymbol{G}_j})).$$
5. It randomises each row tag τ_j into $\tau_j^\star$ and adapts the signatures $\sigma'_{j,i}, \Sigma_{\mathsf{PKE},j,i}, \Sigma'_{\boldsymbol{G}_j}$ accordingly, using the same row-tag randomization randomness for all signatures in row j and leaving the column tags unchanged:
$$\begin{aligned}
((\tau_j^\star, \theta_i), \sigma'^\star_{j,i}) &\leftarrow \text{2-tag-}\mathsf{LHS.RandTag}(\mathsf{LHS.vk}, \tau_j, \theta_i, \bar{C}'_{j,i}, \sigma'_{j,i});\\
((\tau_j^\star, \theta_i), \Sigma^\star_{\mathsf{PKE},j,i}) &\leftarrow \text{2-tag-}\mathsf{LHS.RandTag}(\mathsf{LHS.vk}, \tau_j, \theta_i, \bar{C}_{\mathsf{PKE}}, \Sigma_{\mathsf{PKE},j,i});\\
((\tau_j^\star, \theta_{n+1}), \Sigma'^\star_{\boldsymbol{G}_j}) &\leftarrow \text{2-tag-}\mathsf{LHS.RandTag}(\mathsf{LHS.vk}, \tau_j, \theta_{n+1}, \boldsymbol{G}'_j, \Sigma'_{\boldsymbol{G}_j}).
\end{aligned}$$
6. It samples a random permutation $\Pi : [n] \to [n]$ and assigns
$$\begin{aligned}
&C^{(s)}_{\Pi(j),i} = C'_{j,i},\ \sigma^{(s)}_{\Pi(j),i} = \sigma'^\star_{j,i},\ \Sigma^{(s)}_{\mathsf{PKE},\Pi(j),i} = \Sigma^\star_{\mathsf{PKE},j,i},\ \tau^{(s)}_{\Pi(j)} = \tau_j^\star;\\
&G^{(s)}_{1,\Pi(j)} = G'_{1,j};\ \Sigma^{(s)}_{\boldsymbol{G}_{\Pi}(j)} = \Sigma'^\star_{\boldsymbol{G}_j};\ G^{(s)}_0 = G'_0
\end{aligned}$$
7. It computes $\mathsf{proof}^{(s)} \leftarrow \mathsf{NIZK}_{\mathsf{DH}}.\mathsf{Prove}\left(G_0, G_0^{(s)}, \sum_{j\in[n]} G_{1,j}, \sum_{j\in[n]} G^{(s)}_{1,j}; \beta\right)$
8. It signs the proofs as $\mathsf{sig}^{(s)}_{\mathsf{mix}} \leftarrow \mathsf{Sig.Sign}(\mathsf{SK}_s^{\mathsf{mix}}, \mathsf{proof}^{(s)})$.

The output consists of the box

$$\mathsf{MBox}^{(s)} = \{(C^{(s)}_{j,i}, \sigma^{(s)}_{j,i}, \Sigma^{(s)}_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau^{(s)}_j, G^{(s)}_{1,j}, \Sigma^{(s)}_{\boldsymbol{G}_j})_{j\in[n]}, (\theta_i)_{i\in[n+1]}, G^{(s)}_0, C_{\mathsf{PKE}}\}$$

together with $\mathsf{proof}^{(s)}$ and $\mathsf{sig}^{(s)}_{\mathsf{mix}}$.

Fig. 14: Shuffle and re-randomisation of OMIX.

$\mathsf{OMIX.VerifShuffle}(\mathsf{MBox}^{(0)}, \mathsf{MBox}^{(N_{\mathsf{mix}})}, (\mathsf{proof}^{(s)}, \mathsf{sig}^{(s)}_{\mathsf{mix}}, \mathsf{VK}^{\mathsf{mix}}_s)_{s\in[N_{\mathsf{mix}}]})$:

The verification continues unless the inputs $\mathsf{MBox}^{(0)}, \mathsf{MBox}^{(N_{\mathsf{mix}})}$ cannot be parsed in the following form

$$\mathsf{MBox}^{(0)} = \{(C^{(0)}_{j,i}, \sigma^{(0)}_{j,i}, \Sigma^{(0)}_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau^{(0)}_j, G^{(0)}_{1,j}, \Sigma^{(0)}_{\boldsymbol{G}_j})_{j\in[n]}, (\theta_i)_{i\in[n+1]}, G^{(0)}_0, C_{\mathsf{PKE}}\};$$

$$\mathsf{MBox}^{(N_{\mathsf{mix}})} = \{(C^{(N_{\mathsf{mix}})}_{j,i}, \sigma^{(N_{\mathsf{mix}})}_{j,i}, \Sigma^{(N_{\mathsf{mix}})}_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau^{(N_{\mathsf{mix}})}_j, G^{(N_{\mathsf{mix}})}_{1,j}, \Sigma^{(N_{\mathsf{mix}})}_{\boldsymbol{G}_j})_{j\in[n]},$$
$$(\theta_i)_{i\in[n+1]}, G^{(N_{\mathsf{mix}})}_0, C_{\mathsf{PKE}}\}.$$

The verification outputs 1 if:

- $\mathsf{Sig.Verify}(\mathsf{VK}^{\mathsf{mix}}_s, \mathsf{proof}^{(s)}, \mathsf{sig}^{(s)}_{\mathsf{mix}}) = 1$ for each $s \in [N_{\mathsf{mix}}]$[a];
- $\mathsf{NIZK}_{\mathsf{DH}}.\mathsf{Verify}(\mathsf{proof}^{(s)}) = 1$ for each $s \in [N_{\mathsf{mix}}]$[b];
- For $N \in \{0, N_{\mathsf{mix}}\}$:
  - for each (j, i), parse $\bar{C}^{(N)}_{j,i} = ([\![1]\!]_1, C^{(N)}_{j,i})$ then

$$\text{2-tag-}\mathsf{LHS.Verify}(\mathsf{LHS.vk}, (\tau^{(N)}_j, \theta_i), \bar{C}^{(N)}_{j,i}, \sigma^{(N)}_{j,i}) = 1$$

  - for each j, parse $\boldsymbol{G}^{(N)}_j = (G^{(N)}_0, G^{(N)}_{1,j})$, then

$$\text{2-tag-}\mathsf{LHS.Verify}(\mathsf{LHS.vk}, (\tau^{(N)}_j, \theta_{n+1}), \boldsymbol{G}^{(N)}_j, \Sigma^{(N)}_{\boldsymbol{G}_j}) = 1$$

[a] These N_{mix} signatures are batched in parallel using a multi-signature.
[b] These N_{mix} proofs are batched in parallel using a malleable NIZK system.

Fig. 15: Verification of OMIX.

permutation. This certification occurs only in the offline phase of voting; voters are not required to interact with the authority each time they vote.

The protocol's soundness is determined by a family of authorized transformation sets, specified by the protocol itself to define which operations the mix servers are allowed to perform on the certified inputs in the initial mix box.

We also define unlinkability through the following privacy game: the adversary chooses two transformation sets, and the challenger selects a transformation from one of them. The adversary must guess which set contained the applied transformation.

Definition 28 (Unlinkability). *An offline $\mathcal{P}_{OFF}$ for a family of authorized transformation sets $\{\mathcal{T}_{\lambda,n}\}_{\lambda,n}$ is said to provide unlinkability in the certified key setting, if any PPT adversary $\mathcal{A}$ has no more than a negligible advantage in guessing b in the following security game:*

1. *On input a security parameter λ, a number of users n, a number of mix-servers N, the challenger generates public parameters of $\mathcal{P}_{OFF}$ as pp and master keys $(\mathsf{msk}, \mathsf{mpk})$. It provides $(\mathsf{pp}, \mathsf{mpk})$ to $\mathcal{A}$.*
2. *Upon receiving $(\mathsf{pp}, \mathsf{mpk})$, the adversary $\mathcal{A}$ does as follows:*
   - *It determines the set of corrupted users $\mathcal{CS} \subset [n]$ and generates their public keys $(\mathsf{pk}_i)_{i\in\mathcal{CS}}$ itself.*
   - *For $i \in \mathcal{CS}$, it proves knowledge of the secret keys sk_i for pk_i to obtain certifications $\mathcal{CA}.\Sigma_i$, which the challenger generates using msk.*
   - *It determines the set of corrupted mix-servers $\mathcal{J} \subset [N]$ and generates itself their keys $(\mathsf{VK}_s)_{s\in\mathcal{J}}$.*
3. *The challenger randomly chooses a bit $b \xleftarrow{\$} \{0, 1\}$. It generates the keys of the honest users $(\mathsf{sk}_i, \mathsf{pk}_i)_{i\in[n]\setminus\mathcal{CS}}$ and the keys of the honest mix-servers $(\mathsf{SK}_s, \mathsf{VK}_s)_{s\in[N]\setminus\mathcal{J}}$. It certifies their public keys and generates $(\mathcal{CA}.\Sigma_i)_{i\in[n]\setminus\mathcal{CS}}$. The initial mix box is thus defined by $\mathsf{MBox}^{(0)} = (\mathcal{CA}.\Sigma_i)_{i\in[n]}$. The challenger enters into a loop for $s \in [N]$ with $\mathcal{A}$:*
   - *if $s \in \mathcal{J}$, $\mathcal{A}$ builds itself the new box $\mathsf{MBox}^{(s)}$ with the proof $\mathsf{proof}^{(s)}$;*
   - *if $s \notin \mathcal{J}$, $\mathcal{A}$ provides two transformation sets $\mathcal{T}_{s,0}$ and $\mathcal{T}_{s,1}$ of its choice (typically defined by a polynomial-sized string). Then the challenger applies a transformation T from $\mathcal{T}_{s,b}$ to get $\mathsf{MBox}^{(s)} = T(\mathsf{MBox}^{(s-1)})$, and provides the proof $\mathsf{proof}^{(s)}$.*

   In the end, the adversary outputs its guess b' for b. The experiment outputs 1 if $b' = b$ and 0 otherwise.

Although the unlinkability notions for offline protocols such as OMIX and OPAD serve only as auxiliary definitions and do not by themselves guarantee modular security when these protocols are integrated into our voting framework, their corresponding proofs capture key arguments reused in the final analysis in Section 7.4. Presenting these arguments separately modularizes the technical steps and keeps the overall security proof of the voting protocol from becoming overly intricate.

7.2 Deferred Proofs for OMIX

We prove soundness and unlinkability below.

Theorem 7 (OMIX Soundness). *Our OMIX protocol is designed for the family of authorized transformation sets $\mathcal{T}_{\lambda,n}$ such that each transformation $T \in \mathcal{T}_{\lambda,n}$ is specified by a permutation on the set of rows $\Pi : [n] \to [n]$ and randomness $(\gamma_{\mathsf{PKE},j,i})_{j,i\in[n]}$ such that, if $\mathsf{MBox}' = T(\mathsf{MBox})$, then*

$$C'_{\Pi(j),i} = C_{j,i} + \gamma_{\mathsf{PKE},j,i} \cdot C_{\mathsf{PKE}}$$

for all $i, j \in [n]$, where $(C_{j,i})_{j,i\in[n]}$ and C_{PKE} are in MBox and $(C'_{j,i})_{j,i\in[n]}$ are in MBox'. The protocol is sound in the random oracle model under the tag-binding unforgeability property of the 2-tag-LHS scheme, the soundness of the $\mathsf{NIZK}_{\mathsf{DH}}$ system, and the discrete logarithm assumption in the group $\mathbb{G}_1$.

Proof. Since verification accepts, we can parse

$$\mathsf{MBox} = \{(C_{j,i}, \sigma_{j,i}, \Sigma_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau_j, G_{1,j}, \Sigma_{\boldsymbol{G}_j})_{j\in[n]}, (\theta_i)_{i\in[n+1]}, G_0, C_{\mathsf{PKE}}\};$$
$$\mathsf{MBox}' = \{(C'_{j,i}, \sigma'_{j,i}, \Sigma'_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau'_j, G'_{1,j}, \Sigma'_{\boldsymbol{G}_j})_{j\in[n]}, (\theta_i)_{i\in[n+1]}, G'_0, C_{\mathsf{PKE}}\};$$

along with the signed proof $(\mathsf{proof}, \mathsf{sig}_{\mathsf{mix}})$. When the hash function H is modeled as a random oracle $\mathcal{RO}$, with overwhelming probability, the challenger in the soundness game can answer the query $\mathcal{RO}(0)$ with $G_0 \xleftarrow{\$} \mathbb{G}_1$ and the query $\mathcal{RO}(j)$ with $G_{1,j} := \alpha_j \cdot G_0$, where $(\alpha_j)_{j\in[n]}$ are pairwise-distinct random values. The soundness follows from the following verifications:

1. $\mathsf{Sig.Verify}(\mathsf{VK}^{\mathsf{mix}}, \mathsf{proof}, \mathsf{sig}_{\mathsf{mix}}) = 1$ guarantees that the mixing process is executed by the shuffling server that owns $\mathsf{SK}^{\mathsf{mix}16}$;
2. For each $(j, i) \in [n] \times [n]$, one has

   $$\text{2-tag-}\mathsf{LHS.Verify}\left(\mathsf{LHS.vk}, (\tau'_j, \theta_i), ([\![1]\!]_1, C'_{j,i}), \sigma'_{j,i}\right) = 1.$$

   By the tag-binding unforgeability of the 2-tag-LHS scheme, there exists a tag τ_{k_j} in the initial MBox such that

   $$([\![1]\!]_1, C'_{j,i}) = \phi_{k_j,i} \cdot ([\![1]\!]_1, C_{k_j,i}) + \gamma_{\mathsf{PKE},k_j,i} \cdot ([\![0]\!]_1, C_{\mathsf{PKE}})$$

   which implies that $\phi_{k_j,i} = 1$ and then $C'_{j,i} = C_{k_j,i} + \gamma_{\mathsf{PKE},k_j,i} \cdot C_{\mathsf{PKE}}$. It remains to prove that $\{k_j\}_{j\in[n]} = \{j\}_{j\in[n]}$.
3. For each $j \in [n]$, one has

   $$\text{2-tag-}\mathsf{LHS.Verify}\left(\mathsf{LHS.vk}, (\tau'_j, \theta_{n+1}), (G'_0, G'_{1,j}), \Sigma'_{\boldsymbol{G}_j}\right) = 1.$$

   Also by the tag-binding unforgeability, this implies that $(G'_0, G'_{1,j}) = (\beta \cdot G_0, \beta \cdot G_{1,k_j})$ for some β.
4. A valid proof ensures the same β and that

   $$G'_0 = \beta \cdot G_0; \qquad \sum_{j\in[n]} G'_{1,j} = \beta \cdot \sum_{j\in[n]} G_{1,j}.$$

   Then one has

   $$\sum_{j\in[n]} G_{1,k_j} = \sum_{j\in[n]} G_{1,j}.$$

   If $\{k_j\}_{j\in[n]} \neq \{j\}_{j\in[n]}$, one can use this game to solve a DLOG instance that is embedded by guessing as $G_{1,j^\star}$, in which a correct guess means that $G_{1,j^\star}$ does not appear in $\{G_{1,k_j}\}_{j\in[n]}$.

Therefore, with an overwhelming probability, one has $\{k_j\}_{j\in[n]} = \{j\}_{j\in[n]}$, which implies that there exists a permutation $\Pi : [n] \to [n]$ such that $C'_{\Pi(j),i} = C_{j,i} + \gamma_{\mathsf{PKE},j,i} \cdot C_{\mathsf{PKE}}$. □

The following remarks give two methods for a challenger in the OMIX protocol to extract the permutation applied by an adversary acting as a corrupted mix-server. These methods can be interleaved during the unlinkability-proof simulation, keeping honest servers' permutations private while making corrupted servers' permutations extractable. This strong security property is essential to proving ballot privacy in an OMIX-based voting system with multiple (possibly corrupted) mix-servers.

[16] In case the adversary corrupts one of multiple servers, the mix is identified.

Remark 5 (Permutation Extractor with $\mathcal{RO}$). In the soundness proof, using the values $(\alpha_j)_{j\in[n]}$ assigned to the $\mathcal{RO}$ queries on inputs $(j)_{j\in[n]}$, the challenger can ensure the knowledge soundness of the mix by extracting the permutation applied by the adversary. Furthermore, this knowledge soundness does not rely on the $\mathsf{NIZK}_{\mathsf{DH}}$ proof system. The extraction proceeds as follows:

- Under the soundness of the verifications (with 2-tag-LHS and a standard signature) at points 1, 2, and 3, the challenger has
$$(G'_0, G'_{1,j}) = (\beta \cdot G_0, \beta \cdot G_{1,k_j})$$
for some β, some k_j and all $j \in [n]$.
- For each j, it searches for α_{κ_j} with $\kappa_j \in [n]$ such that $\alpha_{\kappa_j} \cdot G'_0 = G'_{1,j}$. If so, then $k_j = \kappa_j$, and the challenger stores (j, κ_j). Otherwise, the mix is not correct.
- It tests whether $\{\kappa_j\}_{j\in[n]} = \{j\}_{j\in[n]}$. If not, the mix is not correct. If so, this implies $\{k_j\}_{j\in[n]} = \{j\}_{j\in[n]}$, and the challenger then knows the permutation $\Pi : k_j \to j$ that was applied by the adversary.

To distinguish these exponents from the row (verification) keys, which are hash outputs in $\mathbb{G}_1$, we call $(\alpha_j)_{j\in[n]}$ the row secret keys.

Remark 6 (Permutation Extractor with Decryption Key). In the soundness proof, using the decryption key $\mathsf{PKE.sk}$ of the PKE scheme and the extracted secret keys sk_i of the corrupted voters, the challenger can ensure the knowledge soundness of the mix by extracting the permutation applied by the adversary. Furthermore, this knowledge soundness does not rely on the $\mathsf{NIZK}_{\mathsf{DH}}$ proof system. The extraction proceeds as follows:

- Given the encrypted functional keys $(\mathsf{sk}_i = \mathsf{ip.dk}^i_{(1,0)}, \mathsf{ip.dk}^i_{(0,0)})$[17] submitted as mix inputs by the voters, the challenger has
$$C'_{j,i} = C_{k_j,i} + \gamma_{\mathsf{PKE},k_j,i} \cdot C_{\mathsf{PKE}}$$
under the soundness of the verifications (with 2-tag-LHS and a standard signature) at points 1, 2, and 3.
- The challenger decrypts each $C'_{j,i}$ into $m'_{j,i}$. Then one has
$$m'_{j,i} := \mathsf{PKE.Dec}(\mathsf{PKE.sk}, C_{k_j,i}),$$
which means $m'_{j,i} = \mathsf{ip.dk}^i_{(1,0)}$ if $k_j = i$ and $m'_{j,i} = \mathsf{ip.dk}^i_{(0,0)}$ if $k_j \neq i$.
- By decrypting
$$\{C'_{j,i}\}_{j,i\in[n]} \quad \text{into} \quad \{m'_{j,i}\}_{j,i\in[n]},$$
the challenger can test whether $\boldsymbol{M} = \{m'_{j,i}\}_{j,i\in[n]}$ is a matrix resulting from applying some permutation $\Pi : k_j \to j$ to the rows of the identity matrix with which the functional keys are associated during the certification phase. If not, the mix is not correct. If so, this implies that Π was applied by the adversary.

Theorem 8 (OMIX Unlinkability). *Our OMIX protocol is designed for the family of authorized transformation sets $\mathcal{T}_{\lambda,n}$ such that each $T \in \mathcal{T}_{\lambda,n}$ is specified by a permutation on the set of rows $\Pi : [n] \to [n]$ and scalars $\gamma_{\mathsf{PKE},j,i} \in \mathbb{Z}_p$ such that if $\mathsf{MBox}' = T(\mathsf{MBox})$ then*

$$C'_{\Pi(j),i} = C_{j,i} + \gamma_{\mathsf{PKE},j,i} \cdot C_{\mathsf{PKE}}$$

for all $i, j \in [n]$ where $C_{j,i}$ and C_{PKE} are in MBox and $C'_{\Pi(j),i}$ are in MBox'. The challenge pairs of transformation sets are $\mathcal{T}_{s,0}$ and $\mathcal{T}_{s,1}$ that correspond to transformations specified by permutations $\Pi_{s,0}$ and $\Pi_{s,1}$ respectively. The protocol then has unlinkability in the random oracle model under the DDH assumption in the group $\mathbb{G}_1$, the tag randomizability of the 2-tag-LHS scheme and the zero-knowledge property of the $\mathsf{NIZK}_{\mathsf{DH}}$ system.

Proof. We prove this theorem through the following sequence of games, which differ in the $\mathsf{OMIX.Shuffle}$ algorithm as described in Figure 16. As in the construction, the single randomiser C_{PKE} is shared by all entries and remains the last component of every mix box.

Game $\mathbf{G}_0$: This is the real game, as in Definition 28. The security game is specified by the protocol as follows:

[17] Note that $\mathsf{ip.dk}^i_{(1,0)}$ and $\mathsf{ip.dk}^i_{(0,0)}$ might not be valid functional keys but must differ.

- The challenger runs the $\mathsf{OMIX.SetUp}$ algorithm to obtain public parameters pp and generates the signing key $(\mathsf{LHS.sk}, \mathsf{LHS.vk})$ in $\mathsf{OMIX.KeyGen}$. It stores $\mathsf{msk} = \mathsf{LHS.sk}$ and passes pp and $\mathsf{mpk} = \mathsf{LHS.vk}$ to $\mathcal{A}$. In the random oracle model, with overwhelming probability, the challenger can answer the query $\mathcal{RO}(0)$ with $G_0 \xleftarrow{\$} \mathbb{G}_1$ and the query $\mathcal{RO}(j)$ with $G_{1,j} := \alpha_j \cdot G_0$, where $(\alpha_j)_{j\in[n]}$ are pairwise-distinct random values.
- The adversary chooses a set of corrupted users $\mathcal{CS} \subset [n]$ and a set of corrupted mix-servers $\mathcal{J} \subset [N]$. Following the proof of knowledge required for certification, the challenger receives the following keys:
$$\mathsf{sk}_i := \mathsf{ip.dk}^i_{(1,0)};\ \mathsf{pk}_i := (\mathsf{id}_i, \mathsf{ct}_i, \mathsf{ip.dk}^i_{(0,1)}, \mathsf{ip.dk}^i_{(0,0)}, \pi^{\mathsf{pk}}_i)$$
and keys for the corrupted mix-servers $(\mathsf{SK}^{\mathsf{mix}}_s)_{s\in\mathcal{J}}$.
- The challenger chooses a bit $b \xleftarrow{\$} \{0,1\}$. It generates keys for honest users $(\mathsf{pk}_i, \mathsf{sk}_i)_{i\in[n]\setminus\mathcal{CS}}$ and keys for honest mix-servers $(\mathsf{SK}^{\mathsf{mix}}_s, \mathsf{VK}^{\mathsf{mix}}_s)_{s\in[N]\setminus\mathcal{J}}$. It outputs the initial mix box $\mathsf{MBox}^{(0)}$ as
$$\{(C^{(0)}_{j,i}, \sigma^{(0)}_{j,i}, \Sigma^{(0)}_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau^{(0)}_j, G^{(0)}_{1,j}, \Sigma^{(0)}_{\boldsymbol{G}_j})_{j\in[n]}, (\theta_i)_{i\in[n+1]}, G^{(0)}_0, C_{\mathsf{PKE}}\}$$
by running $\mathsf{OMIX.Certify}(\mathsf{msk}, (\mathsf{pk}_i)_{i\in[n]}, (\mathsf{VK}^{\mathsf{mix}}_s)_{s\in[N]})$.
- The challenger enters into a loop for $s \in [N]$ with $\mathcal{A}$:
  - if $s \in \mathcal{J}$, $\mathcal{A}$ builds the new box $\mathsf{MBox}^{(s)}$ itself and provides the proof $\mathsf{proof}^{(s)}$ and signature $\mathsf{sig}^{(s)}$;
  - if $s \notin \mathcal{J}$, when $\mathcal{A}$ provides two permutations over the set of rows, $\Pi_{s,0}$ and $\Pi_{s,1}$, the challenger transforms the mix box $\mathsf{MBox}^{(s-1)}$ into $\mathsf{MBox}^{(s)}$ using $\Pi_{s,b}$ as in the
$$\mathsf{OMIX.Shuffle}(\mathsf{SK}^{\mathsf{mix}}_s, \mathsf{MBox}^{(s-1)}, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)}_{\mathsf{mix}})_{k\in[s-1]})$$
algorithm.

Game $\mathbf{G}_1$: In this game, we switch the $\mathsf{NIZK}_{\mathsf{DH}}$ proof system for
$$(G_0, G^{(s)}_0, \sum_{j\in[n]} G_{1,j}, \sum_{j\in[n]} G^{(s)}_{1,j})$$
to the zero-knowledge setting (proving without knowing β): the permutation over the set of rows can be tested and extracted using either the row secret keys or the PKE decryption key, as presented in Remark 5 and Remark 6. Any adversary that can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ distinguishes between the real and simulated zero-knowledge proofs. The input box to the honest server considered in Lemma 2 is thus correctly transformed from the initial box with overwhelming probability.

Game $\mathbf{G}_2$: In this game, all the honest mix-servers generate random row tags $(\tau^\star_j)_{j\in[n]}$, random encryptions $(C'_{j,i})_{j,i\in[n]}$ of $\mathbf{0}$, for all the users, random row verification keys $(G'_0, (G'_{1,j})_{j\in[n]})$ and generate the signatures using the signing key $\mathsf{LHS.sk}$. Then, they apply the permutation $\Pi_{s,b}$ on the set of rows. In this last game, $\Pi_{s,b}$ is applied on the rows of random encryptions, random row verification keys, and random row tags. The resulting box is then identical to the one from applying $\Pi_{s,1-b}$, which implies that the advantage of any adversary in this game is 0.

We prove the theorem using the following lemma.

Lemma 2. *Under the DDH assumption in $\mathbb{G}_1$, the adversarial views in $\mathbf{G}_1$ and $\mathbf{G}_2$ are computationally indistinguishable.*

In the next hybrid games, from $\mathbf{G}_{1.0}(s_l)$ to $\mathbf{G}_{1.3}(s_l)$, for $l = N - |J|, \ldots, 1$, each mix-server s_l is honest, and the input ballots in $\mathsf{MBox}^{(s_l-1)}$ contain properly permuted randomized ciphertexts from the initial box, as nothing is modified before $\mathbf{G}_1$ except for the zero-knowledge setting. In the following series of hybrid games, for index s_l, the honest mix-servers up to the $(s_l - 1)$-th round play as in $\mathbf{G}_1$, and from the $(s_l + 1)$-th round onward, they play as in $\mathbf{G}_2$. Only the behaviour of the s_l-th mix-server is modified, starting from an honest behavior. Hence, $\mathbf{G}_{1.0}(s_{N-|J|}) = \mathbf{G}_1$ and $\mathbf{G}_{1.3}(s_1) = \mathbf{G}_2$.

Game $\mathbf{G}_{1.0}(s_l)$: In this game, we assume that the initial ballot-box has been correctly generated, and mixing steps up to $\mathsf{MBox}^{(s_l)}$ have been honestly generated (except the zero-knowledge proofs that have been simulated). The next rounds are generated as in $\mathbf{G}_2$.

Game $\mathbf{G}_{1.1}(s_l)$: We modify the signature adaptation and the randomization of the row tags by the s_l-th mix-server. The mixing step in $\mathbf{G}_{1.0}(l)$ consists of updating the ciphertexts, adapting their signatures accordingly, and then re-randomizing the row tags from $(\tau_j)_{j\in[n]}$ into $(\tau_j^\star)_{j\in[n]}$. In this game, the s_l-th honest mix-server uses $\mathsf{LHS.sk}$ to sign the updated ciphertexts directly under freshly generated row tags $(\tau_j^\star)_{j\in[n]}$. The indistinguishability between $\mathbf{G}_{1.0}(s_l)$ and $\mathbf{G}_{1.1}(s_l)$ is perfect, thanks to the $\mathcal{T}_1$-Tag Unlinkability in the 2-tag-LHS scheme.

Game $\mathbf{G}_{1.2}(s_l)$: We modify the randomization of the row keys $(G_0, (G_{1,j})_{j\in[n]})$ by the s_l-th mix-server. The mixing step in $\mathbf{G}_{1.1}(s_l)$ consists of updating $(G_0, (G_{1,j})_{j\in[n]})$ to $(G'_0 = \beta\cdot G_0, (G'_{1,j} = \beta\cdot G_{1,j})_{j\in[n]})$ for some random β. In this game, the s_l-th honest mix-server randomly generates $G'_0 \xleftarrow{\$} \mathbb{G}_1$ and $G'_{1,j} \xleftarrow{\$} \mathbb{G}_1$ for all $j \in [n]$. We note that the Diffie-Hellman proof system is already in the zero-knowledge setting in this game; therefore, generating a valid proof without knowing the factor β is feasible.
The indistinguishability between $\mathbf{G}_{1.1}(s_l)$ and $\mathbf{G}_{1.2}(s_l)$ is implied by the Multi-DDH Assumption (given in Definition 3), where the adversary receiving an instance $(X, (Y_j, Z_j)_{j\in[n]})$ can play as the unlinkability challenger and embed the initial row keys in $\mathsf{MBox}^{(0)}$ as $(G_0, (G_{1,j} := Y_j)_{j\in[n]})$ and the output row keys by the s_l-th mix-server as $(G'_0 := X, (G'_{1,j} := Z_j)_{j\in[n]})$.
During the above simulation, the challenger uses the PKE decryption key to verify the outputs of the corrupted mix-servers and extract their permutations until the mixing round of the next honest mix-server s_{l+1}. Starting from the round of s_{l+1}, the honest mix-servers will play as in $\mathbf{G}_2$, so extracting the corrupted permutations using the freshly generated row secret keys is feasible.

Game $\mathbf{G}_{1.3}(s_l)$: We modify the randomization of the encryptions $(C_{j,i})_{j,i\in[n]}$ by the s_l-th mix-server. The mixing step in $\mathbf{G}_{1.2}(s_l)$ consists of updating $(C_{j,i})_{j,i\in[n]}$ to $(C'_{j,i} := C_{j,i} + \gamma_{\mathsf{PKE},j,i}\cdot C_{\mathsf{PKE}})_{j,i\in[n]}$ for some random factors $(\gamma_{\mathsf{PKE},j,i})_{j,i\in[n]}$. In this game, the s_l-th honest mix-server generates

$$C'_{j,i} = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathbf{0}; r'_{j,i})$$

for all $j, i \in [n]$. As long as the re-randomization is equivalent to a re-encryption, the indistinguishability between $\mathbf{G}_{1.2}(s_l)$ and $\mathbf{G}_{1.3}(s_l)$ is implied by the IND-CPA security of the underlying PKE scheme. During this simulation, the challenger uses the row secret keys to verify and extract corrupted permutations.

As the proof of the lemma is complete, the proof of the theorem is also complete. □

Corollary 1 (OMIX Strong Unlinkability). *The unlinkability of the OMIX protocol holds even if, prior to outputting its guess on the challenge bit (as defined in Definition 28), the adversary $\mathcal{A}$ is provided with an oracle $\mathsf{OCorPermute}$ that returns the permutations Π_s applied by the corrupted servers $s \in \mathcal{J}$ during the mixing loop with the challenger.*

Proof. The corollary follows directly from the proof of Theorem 8. Specifically, the challenger returns the corrupted permutations it has extracted by interleaving the extraction techniques described in Remark 5 and Remark 6. □

7.3 OPAD for Randomness Generation

We provide a construction for OPAD with multiple servers in Figure 17 and prove its soundness and unlinkability within the offline-protocol framework.

Theorem 9 (OPAD Soundness). *Our OPAD protocol is designed for the family of authorized transformation sets $\mathcal{T}_{\lambda,n}$ such that each $T \in \mathcal{T}_{\lambda,n}$ is specified by one-time pads $(z_j)_{j\in[n]}$ and key randomness $(\gamma_{\mathsf{IPE},j,i})_{j,i\in[n]}$, along with a set of implicit PKE randomness, such that if $\mathsf{OBox}' = T(\mathsf{OBox})$, then $\mathsf{OBox}' = \mathsf{OBox}||(C_{j,i})_{j,i\in[n]}, (\mathsf{ct}_j^{\mathsf{PAD}})_{j\in[n]}$, where*

$$C_{j,i} = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \gamma_{\mathsf{IPE},j,i}\cdot \mathsf{ip.dk}^i_{(0,0)} + z_j \cdot \mathsf{ip.dk}^i_{(0,1)})$$

and

$$\mathsf{ct}_j^{\mathsf{PAD}} = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, [\![z_j]\!]_1)\ \forall j \in [n]$$

for all $i, j \in [n]$. In this protocol, $\mathsf{PKE.pk}$ is given as a public parameter, and $(\mathsf{ip.dk}^i_{(0,0)}, \mathsf{ip.dk}^i_{(0,1)})$ are given as part of the user public key. The protocol is sound if the $\mathsf{NIZK}_{\mathsf{Gen}}$ system is sound.

$\mathbf{G}_0$, $\mathbf{G}_1$, $\mathbf{G}_{1.1}(s_l)$, $\mathbf{G}_{1.2}(s_l)$, $\mathbf{G}_{1.3}(s_l)$

$\mathsf{OMIX.Shuffle}(\mathsf{SK}^{\mathsf{mix}}_{s_l}, \mathsf{MBox}^{(s_l-1)}, (\mathsf{proof}^{(k)}, \mathsf{sig}^{(k)}_{\mathsf{mix}})_{k\in[s_l-1]})$[a]:

Each server $\mathcal{S}_s$ takes input $\mathsf{MBox}^{(s_l-1)}$ as

$$\{(C_{j,i}, \sigma_{j,i}, \Sigma_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau_j, G_{1,j}, \Sigma_{\boldsymbol{G}_j})_{j\in[n]}, (\theta_i)_{i\in[n+1]}, G_0, C_{\mathsf{PKE}}\}$$

1. It generates random $\gamma_{\mathsf{PKE},j,i}$ and randomises $C_{j,i}$ for each (j,i):

$$C'_{j,i} = C_{j,i} + \gamma_{\mathsf{PKE},j,i} \cdot C_{\mathsf{PKE}} \qquad C'_{j,i} = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathbf{0}; r'_{j,i})$$

2. It adapts each signature $\sigma_{j,i}$ into $\sigma'_{j,i}$ for $\bar{C}'_{j,i} = ([\![1]\!]_1, C'_{j,i})$ accordingly

$$\sigma'_{j,i} \leftarrow \text{2-tag-}\mathsf{LHS.DerivSign}\big(\mathsf{LHS.vk}, (\tau_j, \theta_i), (\gamma_{\mathsf{PKE},j,i}, \bar{C}_{\mathsf{PKE}}, \Sigma_{\mathsf{PKE},j,i}), (1, \bar{C}_{j,i}, \sigma_{j,i})\big).$$

This step is omitted

3. It generates random β and randomises G_0 and $G_{1,j}$ for each j:

$$G'_0 = \beta \cdot G_0; \qquad G'_{1,j} = \beta \cdot G_{1,j} \qquad G'_0 \xleftarrow{\$} \mathbb{G}_1; \qquad G'_{1,j} \xleftarrow{\$} \mathbb{G}_1$$

4. It adapts each signature $\Sigma_{\boldsymbol{G}_j}$ into $\Sigma'_{\boldsymbol{G}_j}$ for $\boldsymbol{G}'_j = (G'_0, G'_{1,j})$:

$$\Sigma'_{\boldsymbol{G}_j} \leftarrow \text{2-tag-}\mathsf{LHS.DerivSign}(\mathsf{LHS.vk}, (\tau_j, \theta_{n+1}), (\beta, \boldsymbol{G}_j, \Sigma_{\boldsymbol{G}_j})).$$

This step is omitted

[a] The input box $\mathsf{MBox}^{(s_l-1)}$ is verified and the game stops in case of invalidity.

Fig. 16: Modifications in the OMIX.Shuffle algorithm for the unlinkability game.

5. It randomises each row tag τ_j into $\tau^\star_j$ and adapts the signatures $\sigma'_{j,i}, \Sigma_{\mathsf{PKE},j,i}, \Sigma'_{\boldsymbol{G}_j}$ accordingly, using the same row-tag randomization randomness for all signatures in row j and leaving the column tags unchanged:

$$((\tau^\star_j, \theta_i), \sigma'^\star_{j,i}) \leftarrow \text{2-tag-}\mathsf{LHS.RandTag}(\mathsf{LHS.vk}, \tau_j, \theta_i, \bar{C}'_{j,i}, \sigma'_{j,i});$$
$$((\tau^\star_j, \theta_i), \Sigma^\star_{\mathsf{PKE},j,i}) \leftarrow \text{2-tag-}\mathsf{LHS.RandTag}(\mathsf{LHS.vk}, \tau_j, \theta_i, \bar{C}_{\mathsf{PKE}}, \Sigma_{\mathsf{PKE},j,i});$$
$$((\tau^\star_j, \theta_{n+1}), \Sigma^\star_{\boldsymbol{G}_j}) \leftarrow \text{2-tag-}\mathsf{LHS.RandTag}(\mathsf{LHS.vk}, \tau_j, \theta_{n+1}, \boldsymbol{G}'_j, \Sigma'_{\boldsymbol{G}_j});$$
$$(\tau^\star_j, \tilde{\tau}^\star_j) \leftarrow \text{2-tag-}\mathsf{LHS.NewTag}_{\mathcal{T}_1}(\mathsf{LHS.sk})$$
$$\sigma'^\star_{j,i} \leftarrow \text{2-tag-}\mathsf{LHS.Sign}(\mathsf{LHS.sk}, \tilde{\tau}^\star_j, \tilde{\theta}_i, \bar{C}'_{j,i})$$
$$\Sigma^\star_{\mathsf{PKE},j,i} \leftarrow \text{2-tag-}\mathsf{LHS.Sign}(\mathsf{LHS.sk}, \tilde{\tau}^\star_j, \tilde{\theta}_i, \bar{C}_{\mathsf{PKE}})$$
$$\Sigma^\star_{\boldsymbol{G}_j} \leftarrow \text{2-tag-}\mathsf{LHS.Sign}(\mathsf{LHS.sk}, \tilde{\tau}^\star_j, \tilde{\theta}_{n+1}, \boldsymbol{G}'_j)$$

6. It arranges elements by the challenge permutation $\Pi_{s,b} : [n] \to [n]$

$$C^{(s)}_{\Pi(j),i} = C'_{j,i},\ \sigma^{(s)}_{\Pi(j),i} = \sigma'^\star_{j,i},\ \Sigma^{(s)}_{\mathsf{PKE},\Pi(j),i} = \Sigma^\star_{\mathsf{PKE},j,i},\ \tau^{(s)}_{\Pi(j)} = \tau^\star_j;$$
$$G^{(s)}_{1,\Pi(j)} = G'_{1,j};\ \Sigma^{(s)}_{\boldsymbol{G}_{\Pi(j)}} = \Sigma'_{\boldsymbol{G}_j};\ G^{(s)}_0 = G'_0$$

7. It computes $\mathsf{proof}^{(s)} \leftarrow \mathsf{NIZK}_{\mathsf{DH}}.\mathsf{Prove}\left(G_0, G^{(s)}_0, \sum_{j\in[n]} G_{1,j}, \sum_{j\in[n]} G^{(s)}_{1,j}; \beta\right)$

$\mathsf{proof}^{(s)} \leftarrow \mathsf{NIZK}_{\mathsf{DH}}.\mathsf{Sim}\left(G_0, G^{(s)}_0, \sum_{j\in[n]} G_{1,j}, \sum_{j\in[n]} G^{(s)}_{1,j}\right)$

8. It signs the proofs as $\mathsf{sig}^{(s)}_{\mathsf{mix}} \leftarrow \mathsf{Sig.Sign}(\mathsf{SK}^{\mathsf{mix}}_s, \mathsf{proof}^{(s)})$.

The output $\mathsf{MBox}^{(s)} = \{(C^{(s)}_{j,i}, \sigma^{(s)}_{j,i}, \Sigma^{(s)}_{\mathsf{PKE},j,i})_{j,i\in[n]}, (\tau^{(s)}_j, G^{(s)}_{1,j}, \Sigma^{(s)}_{\boldsymbol{G}_j})_{j\in[n]}, (\theta_i)_{i\in[n+1]}, G^{(s)}_0, C_{\mathsf{PKE}}\}$ and $\mathsf{proof}^{(s)}, \mathsf{sig}^{(s)}_{\mathsf{mix}}$.

Fig. 16: Modifications for the unlinkability game (continued).

$\underline{\mathsf{OTP.Setup}(1^\lambda)}$:
$(\mathsf{PKE.pk}, \mathsf{PKE.sk}) \leftarrow \mathsf{PKE.SetUp}(1^\lambda)$; $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{pp} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{SetUp}(1^\lambda)$
$\mathsf{NIZK}_{\mathsf{Gen}}.\mathsf{pp} \leftarrow \mathsf{NIZK}_{\mathsf{Gen}}.\mathsf{SetUp}(1^\lambda)$, where $\mathsf{NIZK}_{\mathsf{Gen}}$ denotes the proof system for $\mathcal{R}_{\mathsf{Gen}}$.
Let $\mathsf{pp} = (\mathsf{PKE.pk}, \mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{pp}, \mathsf{NIZK}_{\mathsf{Gen}}.\mathsf{pp})$.

$\underline{\mathsf{OTP.KeyGen}(\mathsf{pp})}$: Each voter i for $i \in [n]$ prepares keys to be certified as

$$\mathsf{sk}_i := \perp;\ \mathsf{pk}_i := (\mathsf{id}_i, \mathsf{ip.dk}^i_{(0,1)}, \mathsf{ip.dk}^i_{(0,0)})$$

Each one-time pad server $\mathcal{S}^{\mathsf{otp}}_s$ for $s \in [N_{\mathsf{otp}}]$ generates Sig keys $(\mathsf{SK}^{\mathsf{otp}}_s, \mathsf{VK}^{\mathsf{otp}}_s) \leftarrow \mathsf{Sig.SetUp}(1^\lambda)$. The master public/secret key pair $(\mathsf{mpk}, \mathsf{msk})$ is set to be $(\perp, \perp)$[a].

$\underline{\mathsf{OTP.Certify}((\mathsf{pk}_i)_{i\in[n]}, (\mathsf{VK}^{\mathsf{otp}}_s)_{s\in[N_{\mathsf{otp}}]})}$: The authority simply registers servers' verification keys VK_s. The certification to a voter i is $\mathcal{CA}.\Sigma_i = \mathsf{pk}_i$. The initial box is $\mathsf{OBox}^{(0)} = \{\mathcal{CA}.\Sigma_i\}_{i\in[n]}$.

[a] No secret-key-involved action from the certificate authority is required.

Fig. 17: Setup, key generation, certification and one-time-pad generation.

$\underline{\mathsf{OTP.Gen}(\mathsf{SK}^{\mathsf{otp}}_s, \mathsf{OBox}^{(s-1)})}$: Each server $\mathcal{S}^{\mathsf{otp}}_s$ processes as follows:

1. It generates random $\gamma^{(s)}_{\mathsf{IPE},j,i}, z^{(s)}_j$ and ciphertexts

$$\mathsf{ct}^{\mathsf{PAD},(s)}_j \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, [\![z^{(s)}_j]\!]_1)\ \forall j \in [n]$$
$$C^{(s)}_{j,i} \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \gamma^{(s)}_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)} + z^{(s)}_j \cdot \mathsf{ip.dk}^i_{(0,1)})\ \forall j, i \in [n]$$

2. It generates a proof of correct generation for all $j, i \in [n]$:

$$\mathsf{proof}^{(s)}_{j,i} \leftarrow \mathsf{NIZK}_{\mathsf{Gen}}.\mathsf{Prove}(\mathsf{PKE.pk}, \mathsf{ct}^{\mathsf{PAD},(s)}_j, C^{(s)}_{j,i}, \mathsf{ip.dk}^i_{(0,0)}, \mathsf{ip.dk}^i_{(0,1)}; \gamma^{(s)}_{\mathsf{IPE},j,i}, z^{(s)}_j).$$

3. It generates Fiat-Shamir-based proofs of plaintext:

$$\pi^{\mathsf{FS},(s)}_{j,i} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{Prove}(\mathsf{VK}^{\mathsf{otp}}_s, \mathsf{PKE.pk}, C^{(s)}_{j,i}; \gamma^{(s)}_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)} + z^{(s)}_j \cdot \mathsf{ip.dk}^i_{(0,1)}).$$
$$\pi'^{\mathsf{FS},(s)}_j \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{Prove}(\mathsf{VK}^{\mathsf{otp}}_s, \mathsf{PKE.pk}, \mathsf{ct}^{\mathsf{PAD},(s)}_j; [\![z^{(s)}_j]\!]_1).$$

4. It signs the proofs as $\mathsf{sig}^{(s)}_{\mathsf{otp}} \leftarrow \mathsf{Sig.Sign}(\mathsf{SK}^{\mathsf{otp}}_s, (\mathsf{proof}^{(s)}_{j,i}, \pi^{\mathsf{FS},(s)}_{j,i})_{j,i\in[n]}, (\pi'^{\mathsf{FS},(s)}_j)_{j\in[n]})$

The output box $\mathsf{OBox}^{(s)}$ is

$$\mathsf{OBox}^{(s-1)} || (C^{(s)}_{j,i}, \mathsf{proof}^{(s)}_{j,i}, \pi^{\mathsf{FS},(s)}_{j,i})_{j,i\in[n]}, (\mathsf{ct}^{\mathsf{PAD},(s)}_j, \pi'^{\mathsf{FS},(s)}_j)_{j\in[n]}, \mathsf{sig}^{(s)}_{\mathsf{otp}}.$$

$\underline{\mathsf{OTP.Verif}(\mathsf{OBox}^{(0)}, \mathsf{OBox}^{(N_{\mathsf{otp}})})$: The verification continues unless $\mathsf{OBox}^{(0)}$ and $\mathsf{OBox}^{(N_{\mathsf{otp}})}$ cannot be parsed as $\{\mathcal{CA}.\Sigma_i\}_{i\in[n]}$ and $\{\mathcal{CA}.\Sigma_i\}_{i\in[n]} || ((C^{(s)}_{j,i}, \mathsf{proof}^{(s)}_{j,i}, \pi^{\mathsf{FS},(s)}_{j,i})_{j,i\in[n]}, (\mathsf{ct}^{\mathsf{PAD},(s)}_j, \pi'^{\mathsf{FS},(s)}_j)_{j\in[n]}, \mathsf{sig}^{(s)}_{\mathsf{otp}})_{s\in[N_{\mathsf{otp}}]}$, respectively. The verification outputs 1 if:

- $\mathsf{Sig.Verify}(\mathsf{VK}^{\mathsf{otp}}_s, (\mathsf{proof}^{(s)}_{j,i}, \pi^{\mathsf{FS},(s)}_{j,i})_{j,i\in[n]}, (\pi'^{\mathsf{FS},(s)}_j)_{j\in[n]}, \mathsf{sig}^{(s)}_{\mathsf{otp}}) = 1$ for $s \in [N_{\mathsf{otp}}]$;
- $\mathsf{NIZK}_{\mathsf{Gen}}.\mathsf{Verify}(\mathsf{proof}^{(s)}_{j,i}) = \mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{Verify}(\pi^{\mathsf{FS},(s)}_{j,i}) = \mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}.\mathsf{Verify}(\pi'^{\mathsf{FS},(s)}_j) = 1$ for $j, i \in [n]$ and $s \in [N_{\mathsf{otp}}]$.

Fig. 17: One-time-pad generation and verification (continued).

Proof. As the soundness of $\mathsf{NIZK}_{\mathsf{Gen}}$ ensures the correct generation of each $\mathsf{ct}_j^{\mathsf{PAD}}$ and each $C_{j,i}$, the proof is complete. □

Theorem 10 (OPAD Unlinkability). *Our OPAD protocol is designed for the family of authorized transformation sets $\mathcal{T}_{\lambda,n}$ such that each $T \in \mathcal{T}_{\lambda,n}$ is specified by one-time pads $(z_j)_{j\in[n]}$ and key randomness $(\gamma_{\mathsf{IPE},j,i})_{j,i\in[n]}$, along with a set of implicit PKE randomness, such that if $\mathsf{OBox}' = T(\mathsf{OBox})$, then $\mathsf{OBox}' = \mathsf{OBox}||(C_{j,i})_{j,i\in[n]}, (\mathsf{ct}_j^{\mathsf{PAD}})_{j\in[n]}$, where*

$$C_{j,i} = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \gamma_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)} + z_j \cdot \mathsf{ip.dk}^i_{(0,1)})$$

and

$$\mathsf{ct}_j^{\mathsf{PAD}} = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, [\![z_j]\!]_1) \ \forall j \in [n]$$

for all $i, j \in [n]$. In this protocol, $\mathsf{PKE.pk}$ is given as a public parameter, and $(\mathsf{ip.dk}^i_{(0,0)}, \mathsf{ip.dk}^i_{(0,1)})$ are given as part of the user public key. The challenge pairs of transformation sets are $\mathcal{T}_{s,0}$ and $\mathcal{T}_{s,1}$, which correspond to transformations specified by $(z_j^0, \gamma^0_{\mathsf{IPE},j,i})_{j,i\in[n]}$ and $(z_j^1, \gamma^1_{\mathsf{IPE},j,i})_{j,i\in[n]}$, respectively. The protocol then has unlinkability under the IND-CPA security of the PKE scheme and the zero-knowledge properties of the $\mathsf{NIZK}_{\mathsf{Gen}}$ and $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ systems.

Proof. We will prove this theorem using the following series of games.

Game $\mathbf{G}_0$: This is the real game, with the challenge bit b, as defined in Definition 28.
Game $\mathbf{G}_1$: In this game, for all honest servers s, we switch the $\mathsf{NIZK}_{\mathsf{Gen}}$ proofs for

$$(\mathsf{PKE.pk}, \mathsf{ct}_j^{\mathsf{PAD},(s)}, C_{j,i}^{(s)}, \mathsf{ip.dk}^i_{(0,0)}, \mathsf{ip.dk}^i_{(0,1)})_{j,i\in[n]}$$

to the zero-knowledge setting (proving without knowing $\gamma^{(s,b)}_{\mathsf{IPE},j,i}$ and $z_j^{(s,b)}$). Any adversary that can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ is a distinguisher against the zero-knowledge property of $\mathsf{NIZK}_{\mathsf{Gen}}$.
Game $\mathbf{G}_2$: In this game, for all honest servers s, we switch the $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ proofs for

$$(\mathsf{VK}_s^{\mathsf{otp}}, \mathsf{PKE.pk}, C_{j,i}^{(s)})_{j,i\in[n]} \text{ and } (\mathsf{VK}_s^{\mathsf{otp}}, \mathsf{PKE.pk}, \mathsf{ct}_j^{\mathsf{PAD},(s)})_{j\in[n]}$$

to the zero-knowledge setting (proving without knowing $\gamma^{(s,b)}_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)} + z_j^{(s,b)} \cdot \mathsf{ip.dk}^i_{(0,1)}$ and $[\![z_j^{(s,b)}]\!]_1$). Any adversary that can distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ is a distinguisher against the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$.
Game $\mathbf{G}_2(s_l)$: The modification in this game is made by the honest server s_l (with l running from $N-|J|$ down to 1). The server generates ciphertexts $\mathsf{ct}_j^{\mathsf{PAD},(s)}$ and $C_{j,i}^{(s)}$ that encrypt $\mathbf{0}$, instead of $[\![z_j^{(s,b)}]\!]_1$ and $\gamma^{(s,b)}_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)} + z_j^{(s,b)} \cdot \mathsf{ip.dk}^i_{(0,1)}$, respectively. Any adversary that can distinguish between $\mathbf{G}_2(s_{l-1})$ and $\mathbf{G}_2(s_l)$ is an attacker against the IND-CPA security of the PKE scheme.

For each PKE-based transition, fix the unlinkability challenge bit b and denote the adjacent hybrids by H_0 and H_1, before and after the ciphertext replacement, respectively. A reduction $\mathcal{B}$ submits each original plaintext and $\mathbf{0}$ to the PKE encryption oracle and returns the simulated adversary's output bit as its guess α. The PKE challenge bit, distinct from b, selects H_0 or H_1. By Definition 6, the events $\mathrm{IND}_0^{\mathsf{PKE}} = 1$ and $\mathrm{IND}_1^{\mathsf{PKE}} = 0$ both correspond to $\alpha = 0$. Hence,

$$|\Pr[H_0 = 0] - \Pr[H_1 = 0]| = \mathsf{Adv}^{\mathrm{IND}^{\mathsf{PKE}}}(\lambda, \rho, \mathcal{B}),$$

so each transition has negligible distinguishing advantage. □

Corollary 2 (OPAD Strong Unlinkability). *The unlinkability of the OPAD protocol holds even if, prior to outputting its guess on the challenge bit (as defined in Definition 28), the adversary $\mathcal{A}$ is provided with an oracle $\mathsf{OCorPad}$ that decrypts the ciphertexts $\{\mathsf{ct}_j^{\mathsf{PAD},(s)}\}_{j\in[n]}$ and $\{C_{j,i}^{(s)}\}_{j,i\in[n]}$ that are applied by the corrupted servers $s \in \mathcal{J}$ during the box-appending loop with the challenger.*

Proof. For the transitions between $\mathbf{G}_0$ and $\mathbf{G}_1$ and between $\mathbf{G}_1$ and $\mathbf{G}_2$, which rely on the zero-knowledge property of the underlying proof systems, the challenger can simply use the PKE decryption key to decrypt the corrupted servers' ciphertexts $\{\mathsf{ct}_j^{\mathsf{PAD},(s)}\}_{j\in[n],s\in\mathcal{J}}$ and $\{C_{j,i}^{(s)}\}_{j,i\in[n],s\in\mathcal{J}}$. The PKE reduction uses the same output convention as above and returns an independent uniform bit on a simulation or extraction

abort. As in Lemma 1, the distinguishing advantage is bounded by its PKE advantage plus the sum of the abort probabilities in the two PKE worlds.

For the transition between $\mathbf{G}_2(s_{l-1})$ and $\mathbf{G}_2(s_l)$, which relies on the IND-CPA security of the PKE scheme, the challenger can no longer access the decryption key. Instead, it uses the random oracle in the $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ proof system to extract the plaintexts within ciphertexts of all corrupted servers by rewinding the adversary $\mathcal{A}$. On the other hand, the challenger uses the random oracle to simulate the proofs for the ciphertexts $\{\mathsf{ct}_j^{\mathsf{PAD},(s_l)}\}_{j\in[n]}$ and $\{C_{j,i}^{(s_l)}\}_{j,i\in[n]}$ generated by the honest server s_l. As we hardwire each server's identity $\mathsf{VK}_s^{\mathsf{otp}}$ into the $\mathcal{RO}$ query for proofs, an argument similar to that presented in the proof of Lemma 1 shows that there will be no collisions when assigning $\mathcal{RO}$ inputs and outputs for zero-knowledge simulation and plaintext extraction. Hence, the proof is complete. □

7.4 Integrating Offline Protocols in Voting

Having constructed both offline protocols, OPAD and OMIX, we now state a theorem establishing ballot privacy for an OMIX-based voting scheme in the multi-authority setting[18].

Theorem 11. *The OMIX-based voting framework, as described in Section 5.1, when instantiated using the OMIX protocol from Section 6.4 and the OPAD protocol from Section 7.3, is ballot private in the selective and symmetric-key setting, under the condition that at least one OMIX server and one OPAD server are honest, and corrupted servers are fixed before the run of these protocols.*

Proof. At the end of the game, let L be the list of eligible voters, $\mathcal{CS}$ be the set of voters that are corrupted via QCor, and $\mathcal{HS}$ be the subset of honest voters (whose public keys are generated via QNewHon but not corrupted via QCor) that are queried and recorded in QVote_β. In the selective setting, $\mathcal{HS}$, $\mathcal{CS}$, and $\{(\mathsf{id}, v^0, v^1)\}_{\mathsf{id}\in\mathcal{HS}}$ are specified beforehand from the QNewHon, QCor and QVote_β queries. In the symmetric-key setting, any (id, v^0, v^1) vote queries for $\mathsf{id} \in \mathcal{CS}$ must satisfy $v^0 = v^1$.

We also note that in L, there is also a subset of dropout honest voters (honest-but-not recorded by QVote_β queries), that we denote by $\mathcal{DH}$; and a subset of voters that are registered by the adversary via QReqCast queries, that we denote by $\mathcal{RC}$. The four sets $\mathcal{HS}$, $\mathcal{CS}$, $\mathcal{DH}$, $\mathcal{RC}$ are disjoint sets that make up L.

To align with permutations, we assume $n = |\mathsf{L}|$ and index the voters in L using integers from $[n]$. From this point forward, we denote by $\mathcal{HI}$, $\mathcal{CI}$, $\mathcal{DI}$, $\mathcal{RI}$ as the sets of indices in $[n]$ corresponding to the voters in $\mathcal{HS}$, $\mathcal{CS}$, $\mathcal{DH}$, $\mathcal{RC}$ respectively.

In answering the QPKProcess query, the protocol now involves OMIX and OPAD where the challenger plays both as honest servers and the certificate authority, along with servers that are corrupted by the adversary. The experiment will sequentially run OPAD and OMIX as follows:

- At the beginning of OPAD, the adversary $\mathcal{A}$ fixes N_1 as the number of servers, it determines the set of corrupted ones $\mathcal{J}_1 \subset [N_1]$ and generates itself their keys $(\mathsf{VK}_{s_1})_{s_1\in\mathcal{J}_1}$. The challenger generates the keys of the honest servers $(\mathsf{SK}_{s_1}, \mathsf{VK}_{s_1})_{s_1\in[N_1]\setminus\mathcal{J}_1}$. The challenger certifies inputs of voters $(\mathsf{id}_i, \mathsf{ip.dk}^i_{(0,1)}, \mathsf{ip.dk}^i_{(0,0)})$ and generates certificates $(\mathcal{CA}.\Sigma_i^1)_{i\in[n]}$. The initial box is thus defined by $\mathsf{OBox}^{(0)} = (\mathcal{CA}.\Sigma_i^1)_{i\in[n]}$. The challenger enters into a loop for $s_1 \in [N_1]$ with $\mathcal{A}$:
  - if $s_1 \in \mathcal{J}_1$, $\mathcal{A}$ builds itself the new box $\mathsf{OBox}^{(s_1)}$ with the proof $\mathsf{proof}^{(s_1)}$;
  - if $s_1 \notin \mathcal{J}_1$, the challenger builds the new box $\mathsf{OBox}^{(s_1)}$ with the proof $\mathsf{proof}^{(s_1)}$ as honest mix-servers in Figure 17.

  If OPAD returns a valid output, then the game continues. Otherwise it stops.
- At the beginning of OMIX, the adversary $\mathcal{A}$ fixes N_2 as the number of servers, it determines the set of corrupted ones $\mathcal{J}_2 \subset [N_2]$ and generates itself their keys $(\mathsf{VK}_{s_2})_{s_2\in\mathcal{J}_2}$. The challenger generates the keys of the honest servers $(\mathsf{SK}_{s_2}, \mathsf{VK}_{s_2})_{s_2\in[N_2]\setminus\mathcal{J}_2}$. The challenger certifies inputs of voters $(\mathsf{id}_i, \mathsf{ct}_i, \mathsf{ip.dk}^i_{(0,0)})$ and generates certificates $(\mathcal{CA}.\Sigma_i^2)_{i\in[n]}$. The initial box is thus defined by $\mathsf{MBox}^{(0)} = (\mathcal{CA}.\Sigma_i^2)_{i\in[n]}$. The challenger enters into a loop for $s_2 \in [N_2]$ with $\mathcal{A}$:
  - if $s_2 \in \mathcal{J}_2$, $\mathcal{A}$ builds itself the new box $\mathsf{MBox}^{(s_2)}$ with the proof $\mathsf{proof}^{(s_2)}$;
  - if $s_2 \notin \mathcal{J}_2$, the challenger builds the new box $\mathsf{MBox}^{(s_2)}$ with the proof $\mathsf{proof}^{(s_2)}$ as honest servers in Figure 14.

[18] The ElGamal keypair for the PKE scheme can be jointly generated using standard DKG techniques [GJKR07, Wik05].

If OMIX returns a valid output, then the game continues, and stops otherwise.

Assuming that the output of OPAD contains PKE ciphertexts of randomised functional keys $(C^1_{j,i})_{j,i\in[n]} := (\sum_{s_1\in[N_1]} C^{1,s_1}_{j,i})_{j,i\in[n]}$ and the output of OMIX contains PKE ciphertexts of mixed functional keys $(C^2_{j,i})_{j,i\in[n]} := (C^{2,N_2}_{j,i})_{j,i\in[n]}$, the challenger outputs $(\mathsf{ip.dk}^i_j)_{j,i\in[n]}$ to the adversary by performing a PKE decryption on $(C^1_{j,i} + C^2_{j,i})_{j,i\in[n]}$.

Moreover, accesses to the random oracle in each phase are assumed to be separated by a prefix such as "Registration", "OMIX" or "OPAD", in the $\mathcal{RO}$ input. This helps avoid collisions during the extraction of adversarial inputs and the simulation of honest inputs. Furthermore, to avoid nested rewinding in the simulation of PKE decryption, we perform every rewinding of corrupted parties before the OMIX phase begins, namely up to the verification step of OPAD. Our hybrids are inspired by those used in the security proofs of previous protocols in Section 4.2, Section 5.1, Section 6.4, Section 7.3.

Game $\mathbf{G}_0$: This is the real game, as in Definition 14.

Game $\mathbf{G}_1$: In this game, via a hybrid, we switch all the proof systems that ensure universal verifiability, namely those for the relations in $\{\mathcal{R}_{\mathsf{dk}}, \mathcal{R}_{\mathsf{ct}}, \mathsf{OMIX}, \mathsf{OPAD}, \mathcal{R}_{\mathsf{Dec}}, \mathcal{R}_{\mathsf{ck}}\}$, to the zero-knowledge setting by using a simulator to generate proofs. Through a hybrid argument, one can use the adversary $\mathcal{A}$ to construct a distinguisher between the two settings of one of the proof systems.

We first use hybrids to hide inputs $\mathsf{ip.dk}^i_{(1,0)}$ in ct_i of honest voters in $\mathcal{HI}$.

Game $\mathbf{G}_2$: In this game, by relying on the programmability of the random oracle $\mathcal{RO}$, the challenger simulates Σ-protocol proofs π_i^{FS} for knowledge of the plaintexts under ciphertexts ct_i generated via $\mathsf{QNewHonest}$ queries. The indistinguishability between $\mathbf{G}_1$ and $\mathbf{G}_2$ is statistical, as the probability that the adversary correctly guesses the randomness in ct_i, which is generated honestly in the $\mathsf{QNewHonest}$ oracle, and queries it to $\mathcal{RO}$ is negligible.

Game $\mathbf{G}_3$: In this game, the challenger replaces PKE ciphertexts using the following strategy: it first replaces the ciphertexts ct_i queried via $\mathsf{QNewHonest}$ with encryptions of $\mathbf{0}$. It then runs the $\mathsf{PKProcess}$ protocol (including OMIX) on the replaced ciphertexts ct_i (in pk_i) to respond to the $\mathsf{QPKProcess}$ query. However, the challenger outputs the keys $(\mathsf{ip.dk}^i_j)_{j,i\in[n]}$ as if they had been generated without any replacements. The indistinguishability between G_2 and G_3 follows from the IND-CPA security of the PKE scheme. During the simulation of the PKE security game, instead of using the decryption key, the challenger uses the $\mathcal{RO}$-based extractor from $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ to extract the registered keys $\mathsf{ip.dk}^i_{(1,0)}$ of corrupted voters $i \in \mathcal{RC}$ and the randomness $(\gamma^{(s_1)}_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)} + z^{(s_1)}_j \cdot \mathsf{ip.dk}^i_{(0,1)})_{j,i\in[n]}$ of corrupted servers $s_1 \in \mathcal{J}_1$, and uses the $\mathcal{RO}$-based extractor of OMIX (as described in Remark 5) to extract permutations $\Pi^{(s_2)}$ of corrupted servers $s_2 \in \mathcal{J}_2$.

We continue to hide the randomness $(\gamma^{(s_1)}_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)} + z^{(s_1)}_j \cdot \mathsf{ip.dk}^i_{(0,1)})_{j,i\in[n]}$ contributed by each honest server s_1 in OPAD, via the following sequence of hybrids, indexed by s_1 running backwards through $[N_1] \setminus \mathcal{J}_1$:

Game $\mathbf{G}_3^{\mathsf{OPAD}}(s_1)$: The modification in this game is made by the honest server s_1. The server generates ciphertexts $\mathsf{ct}_j^{\mathsf{PAD},(s_1)}$ and $C^{(s_1)}_{j,i}$ that encrypt $\mathbf{0}$ instead of $[\![z^{(s_1)}_j]\!]_1$ and $(\gamma^{(s_1)}_{\mathsf{IPE},j,i} \cdot \mathsf{ip.dk}^i_{(0,0)} + z^{(s_1)}_j \cdot \mathsf{ip.dk}^i_{(0,1)})$, respectively. However, the challenger outputs the keys $(\mathsf{ip.dk}^i_j)_{j,i\in[n]}$ as if they were generated without any replacements. Any adversary that can distinguish between the two games is an attacker against the IND-CPA security of the PKE scheme. During the simulation of the PKE security game, the challenger uses the $\mathcal{RO}$-based extractor from $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{FS}}}$ to extract the corrupted servers' inputs.

We continue to hide the permutation images $\{\Pi^{(s_2)}(i)\}_{i\in\mathcal{HI}}$ contributed by each honest server s_2 in OMIX, via the following sequence of hybrids, indexed by s_2 running backwards through $[N_2] \setminus \mathcal{J}_2$:

Game $\mathbf{G}_{3.0}^{\mathsf{OMIX}}(s_2)$: In this game, we assume that the initial ballot-box has been correctly generated, and mixing steps up to $\mathsf{MBox}^{(s_2)}$ have been honestly generated (except the zero-knowledge proofs that have been simulated). The next honest rounds are generated as follows:
The honest mix-server generates random row tags $(\tau^\star_j)_{j\in[n]}$, random encryptions $(C'_{j,i})_{j\in[n],i\in\mathcal{HI}}$ of $\mathbf{0}$, and random row verification keys $(G'_0, (G'_{1,j})_{j\in[n]})$, and generates the signatures using the signing key $\mathsf{LHS.sk}$. This perfectly hides the images $\{\Pi^{(s_2)}(i)\}_{i\in\mathcal{HI}}$ of any permutation $\Pi^{(s_2)}$ on the set of rows.

Game $\mathbf{G}_{3.1}^{\mathsf{OMIX}}(s_2)$: We modify the signature adaptation and the randomization of the row tags by the s_2-th mix-server. In this game, the s_2-th honest mix-server uses $\mathsf{LHS.sk}$ to sign the updated ciphertexts directly under freshly generated row tags $(\tau^\star_j)_{j\in[n]}$. The indistinguishability is perfect, thanks to the $\mathcal{T}_1$-Tag Unlinkability of the 2-tag-LHS scheme.

Game $\mathbf{G}_{3.2}^{\mathsf{OMIX}}(s_2)$: We modify the randomization of the row keys $(G_0, (G_{1,j})_{j\in[n]})$ by the s_2-th mix-server. In this game, the s_2-th honest mix-server randomly generates $G'_0 \xleftarrow{\$} \mathbb{G}_1$ and $G'_{1,j} \xleftarrow{\$} \mathbb{G}_1$ for all $j \in [n]$. The indistinguishability is implied by the Multi-DDH assumption (given in Definition 3).
During the above reduction, the challenger uses the PKE decryption key to verify and extract the permutations Π of corrupted servers until the mixing round of the next honest mix-server. Starting from this round, the challenger instead extracts them using its freshly generated row secret keys. The extraction methods are described in Remark 5 and Remark 6, respectively.

Game $\mathbf{G}_{3.3}^{\mathsf{OMIX}}(s_2)$: We modify the randomization of the encryptions $(C_{j,i})_{j,i\in[n]}$ by the s_2-th mix-server. In this game, the s_2-th honest mix-server generates

$$C'_{j,i} = \mathsf{PKE.Enc}(\mathsf{PKE.pk}, \mathbf{0}; r'_{j,i})$$

for all $j \in [n], i \in \mathcal{HI}$. As the re-randomization is equivalent to a re-encryption, the indistinguishability is implied by the IND-CPA of the PKE scheme. During this reduction, the challenger uses row secret keys to verify and extract corrupted permutations.

By extracting corrupted inputs of voters and servers, the challenger retains the programmability of jointly contributed components, namely permutations Π, one-time pads $\{z_j\}_{j\in[n]}$, and key randomness $\{\gamma_{\mathsf{IPE},j,i}\}_{j,i\in[n]}$. With this programmability, the proof continues with a sequence of hybrids based on the IND-CPA security of the FH-IPFE scheme.

Let the challenge votes be $(v_i^\beta)_{i\in\mathcal{HI}}$, where $\beta \in \{0,1\}$. There exists a random permutation Π^β over n such that $\Pi^\beta(i) = i$ for $i \in [n] \setminus \mathcal{HI}$ and for any random permutation Π, applying $\Pi \circ \Pi^\beta$ to $(v_i^\beta)_{i\in[n]}$ is equivalent to applying Π to $(v_i^0)_{i\in[n]}$, assuming $v_i^\beta = v_i^0 = v_i^1$ for $i \in [n] \setminus \mathcal{HI}$.

Denote the columns forming the matrices corresponding to $\Pi \circ \Pi^\beta$ and Π as $(\boldsymbol{p}^{\beta,i})_{i\in[n]}$ and $(\boldsymbol{p}^{0,i})_{i\in[n]}$, respectively. Then

$$\sum_{i\in\mathcal{HI}} p_j^{\beta,i} v_i^\beta = \sum_{i\in\mathcal{HI}} p_j^{0,i} v_i^0 \ \forall j \in [n]$$

by Remark 3. Using an argument similar to that in the proof of Theorem 1, we transition from the last game in the OMIX hybrid sequence to the following game, where there is no information about the challenge bit.

Game $\mathbf{G}_4$: In this game, instead of using $(v_i^\beta)_{i\in\mathcal{HI}}$ and $(\boldsymbol{p}^{\beta,i})_{i\in\mathcal{HI}}$, the challenger uses $(v_i^0)_{i\in\mathcal{HI}}$ and $(\boldsymbol{p}^{0,i})_{i\in\mathcal{HI}}$ to respond to QVote_β and QPKProcess queries, respectively. The indistinguishability between $\mathbf{G}_3$ and $\mathbf{G}_4$ is implied by the SXDH assumption in $\mathcal{PG}$ and the IND-CPA security of the FH-IPFE scheme.

The proof is thus complete. □

In addition to ballot privacy, the multi-server OMIX-based voting scheme preserves self-tallying, client-side lightweightness, and corrective fault tolerance. Universal verifiability is also maintained through the soundness properties of the OMIX and OPAD protocols.

References

AAG+25. L. Ackermann, M. Arapinis, P. Georgiou, N. Lamprou, L. Mareková, and T. Zacharias. E-cclesia: Universally composable self-tallying elections over anonymous broadcast. *IACR Communications in Cryptology*, 2(3), 2025.

Adi08. B. Adida. Helios: Web-based open-audit voting. In *USENIX Security 2008: 17th USENIX Security Symposium*, pages 335–348, San Jose, CA, USA, July 28 – August 1, 2008. USENIX Association.

AGT21. S. Agrawal, R. Goyal, and J. Tomida. Multi-party functional encryption. In *TCC 2021: 19th Theory of Cryptography Conference, Part II*, *Lecture Notes in Computer Science* 13043, pages 224–255, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

ANO+25. M. Abe, M. Nanri, M. Ohkubo, O. Perez-Kempner, D. Slamanig, and M. Tibouchi. A certified-input mixnet from two-party mercurial signatures on randomizable ciphertexts. In *ESORICS 2025: 30th European Symposium on Research in Computer Security, Part II*, Lecture Notes in Computer Science, pages 1–21. Springer, Heidelberg, Germany, September 2025.

AW07a. B. Adida and D. Wikström. How to shuffle in public. In *TCC 2007: 4th Theory of Cryptography Conference*, *Lecture Notes in Computer Science* 4392, pages 555–574, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

AW07b. B. Adida and D. Wikström. Offline/online mixing. In *ICALP 2007: 34th International Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science* 4596, pages 484–495, Wroclaw, Poland, July 9–13, 2007. Springer, Heidelberg, Germany.

BCG+15. D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

BDN18. D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology – ASIACRYPT 2018, Part II*, *Lecture Notes in Computer Science* 11273, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

Ben87. J. Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, September 1987.

Ben06. J. Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, EVT'06, page 5, USA, 2006. USENIX Association.

BG12. S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology – EUROCRYPT 2012*, *Lecture Notes in Computer Science* 7237, pages 263–280, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

BJK15. A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In *Advances in Cryptology – ASIACRYPT 2015, Part I*, *Lecture Notes in Computer Science* 9452, pages 470–491, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

BPW12. D. Bernhard, O. Pereira, and B. Warinschi. On necessary and sufficient conditions for private ballot submission. Cryptology ePrint Archive, Paper 2012/236, 2012.

BR93. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC 2011: 8th Theory of Cryptography Conference*, *Lecture Notes in Computer Science* 6597, pages 253–273, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

CCFG16. P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo. BeleniosRF: A non-interactive receipt-free electronic voting scheme. In *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1614–1625, Vienna, Austria, October 24–28, 2016. ACM Press.

CCM08. M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy*, pages 354–368, Oakland, CA, USA, May 18–21, 2008. IEEE Computer Society Press.

CDG+18. J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In *Advances in Cryptology – ASIACRYPT 2018, Part II*, *Lecture Notes in Computer Science* 11273, pages 703–732, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

CF85. J. D. Cohen and M. J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 372–382, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press.

CFL19. V. Cortier, A. Filipiak, and J. Lallemand. BeleniosVS: Secrecy and verifiability against a corrupted voting device. In *CSF 2019: IEEE 32nd Computer Security Foundations Symposium*, pages 367–381, Hoboken, NJ, USA, June 25–28, 2019. IEEE Computer Society Press.

CGG19. V. Cortier, P. Gaudry, and S. Glondu. Belenios: A simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning: Essays Dedicated to Catherine A. Meadows*, pages 214–238. Springer International Publishing, Cham, 2019.

Cha81. D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

CS11. V. Cortier and B. Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *CSF 2011: IEEE 24th Computer Security Foundations Symposium*, pages 297–311, Domaine de l'Abbaye des Vaux de Cernay, France, June 27–29, 2011. IEEE Computer Society Press.

EY23. M. ElSheikh and A. M. Youssef. Dispute-free scalable open vote network using zk-SNARKs. In *Financial Cryptography and Data Security. FC 2022 International Workshops*, pages 499–515. Springer International Publishing, 2023.

EYH25. M. ElSheikh, A. M. Youssef, and M. A. Hasan. Self-tallying e-voting using homomorphic time-lock puzzles and zk-snarks. *IEEE Transactions on Network Science and Engineering*, 12(4):2566–2581, 2025.

FHS19. G. Fuchsbauer, C. Hanser, and D. Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.

FLSZ17. P. Fauzi, H. Lipmaa, J. Siim, and M. Zając. An efficient pairing-based shuffle argument. In *Advances in Cryptology – ASIACRYPT 2017*, pages 97–127, Cham, 2017. Springer International Publishing.

FR22. A. Faonio and L. Russo. Mix-nets from re-randomizable and replayable cca-secure public-key encryption. In *Security and Cryptography for Networks*, pages 172–196, Cham, 2022. Springer International Publishing.

FS01. J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology – CRYPTO 2001*, *Lecture Notes in Computer Science* 2139, pages 368–387, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

GGG+14. S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In *Advances in Cryptology – EUROCRYPT 2014*, *Lecture Notes in Computer Science* 8441, pages 578–602, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

GI08. J. Groth and Y. Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology – EUROCRYPT 2008*, *Lecture Notes in Computer Science* 4965, pages 379–396, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

GJKR07. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

Gro04a. J. Groth. Efficient maximal privacy in boardroom voting and anonymous broadcast. In *FC 2004: 8th International Conference on Financial Cryptography*, *Lecture Notes in Computer Science* 3110, pages 90–104, Key West, USA, February 9–12, 2004. Springer, Heidelberg, Germany.

Gro04b. J. Groth. Evaluating security of voting schemes in the universal composability framework. In *ACNS 04: 2nd International Conference on Applied Cryptography and Network Security*, *Lecture Notes in Computer Science* 3089, pages 46–60, Yellow Mountain, China, June 8–11, 2004. Springer, Heidelberg, Germany.

GS08. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology – EUROCRYPT 2008*, *Lecture Notes in Computer Science* 4965, pages 415–432, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

HM20. T. Haines and J. Müller. SoK: Techniques for verifiable mix nets. In *CSF 2020: IEEE 33rd Computer Security Foundations Symposium*, pages 49–64, Boston, MA, USA, June 22–26, 2020. IEEE Computer Society Press.

HMMP23. T. Haines, R. Mosaheb, J. Müller, and I. Pryvalov. SoK: Secure e-voting with everlasting privacy. *Proceedings on Privacy Enhancing Technologies*, 2023(1):279–293, 2023.

HPP20. C. Hébant, D. H. Phan, and D. Pointcheval. Linearly-homomorphic signatures and scalable mix-nets. In *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, *Lecture Notes in Computer Science* 12111, pages 597–627, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

HRZ10. F. Hao, P. Ryan, and P. Zielinski. Anonymous voting by two-round public discussion. *Information Security, IET*, 4:62–67, 07 2010.

Int25. International Association for Cryptologic Research. Iacr news item: 21 november 2025. `https://www.iacr.org/news/item/27138`, 2025.

JCJ05. A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, page 61–70, New York, NY, USA, 2005. Association for Computing Machinery.

KDK11. K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *PETS 2011: 11th International Symposium on Privacy Enhancing Technologies*, *Lecture Notes in Computer Science* 6794, pages 175–191, Waterloo, ON, Canada, July 27–29, 2011. Springer, Heidelberg, Germany.

KSRH12. D. Khader, B. Smyth, P. Y. A. Ryan, and F. Hao. A fair and robust voting system by broadcast. In *Electronic Voting 2012 (EVOTE 2012)*, *Lecture Notes in Informatics* 205, pages 285–299. Gesellschaft für Informatik, 2012.

KY02. A. Kiayias and M. Yung. Self-tallying elections and perfect ballot secrecy. In *PKC 2002: 5th International Workshop on Theory and Practice in Public Key Cryptography*, *Lecture Notes in Computer Science* 2274, pages 141–158, Paris, France, February 12–14, 2002. Springer, Heidelberg, Germany.

Lin17. H. Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In *Advances in Cryptology – CRYPTO 2017, Part I*, *Lecture Notes in Computer Science* 10401, pages 599–629, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

LSY+22. Y. Li, W. Susilo, G. Yang, Y. Yu, D. Liu, X. Du, and M. Guizani. A blockchain-based self-tallying voting protocol in decentralized iot. *IEEE Transactions on Dependable and Secure Computing*, 19(1):119–130, 2022.

MSH17. P. McCorry, S. F. Shahandashti, and F. Hao. A smart contract for boardroom voting with maximum voter privacy. In *FC 2017: 21st International Conference on Financial Cryptography and Data Security*, *Lecture Notes in Computer Science* 10322, pages 357–375, Sliema, Malta, April 3–7, 2017. Springer, Heidelberg, Germany.

O'N10. A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

PBHR20. S. Panja, S. Bag, F. Hao, and B. Roy. A smart contract system for decentralized borda count voting. *IEEE Transactions on Engineering Management*, 67(4):1323–1339, 2020.

Poi24. D. Pointcheval. Efficient universally-verifiable electronic voting with everlasting privacy. In *Security and Cryptography for Networks*, pages 323–344, Cham, 2024. Springer Nature Switzerland.

PRT12. U. Parampalli, K. Ramchen, and V. Teague. Efficiently shuffling in public. In *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, *Lecture Notes in Computer Science* 7293, pages 431–448, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany.

SK95. K. Sako and J. Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *Advances in Cryptology – EUROCRYPT'95*, *Lecture Notes in Computer Science* 921, pages 393–403, Saint-Malo, France, May 21–25, 1995. Springer, Heidelberg, Germany.

SW05. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In *Advances in Cryptology – EUROCRYPT 2005*, *Lecture Notes in Computer Science* 3494, pages 457–473, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

SW21. E. Shi and K. Wu. Non-interactive anonymous router. In *Advances in Cryptology – EUROCRYPT 2021, Part III*, *Lecture Notes in Computer Science* 12698, pages 489–520, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

TAO16. J. Tomida, M. Abe, and T. Okamoto. Efficient functional encryption for inner-product values with full-hiding security. In *ISC 2016: 19th International Conference on Information Security*, *Lecture Notes in Computer Science* 9866, pages 408–425, Honolulu, HI, USA, September 3–6, 2016. Springer, Heidelberg, Germany.

The18. The State of Geneva. Chvote. `https://github.com/republique-et-canton-de-geneve/chvote-protocol-poc`, 2018.

TW10. B. Terelius and D. Wikström. Proofs of restricted shuffles. In *AFRICACRYPT 10: 3rd International Conference on Cryptology in Africa*, *Lecture Notes in Computer Science* 6055, pages 100–113, Stellenbosch, South Africa, May 3–6, 2010. Springer, Heidelberg, Germany.

Wik05. D. Wikström. Universally composable DKG with linear number of exponentiations. In *SCN 04: 4th International Conference on Security in Communication Networks*, *Lecture Notes in Computer Science* 3352, pages 263–277, Amalfi, Italy, September 8–10, 2005. Springer, Heidelberg, Germany.

Wik09. D. Wikström. A commitment-consistent proof of a shuffle. In *ACISP 09: 14th Australasian Conference on Information Security and Privacy*, *Lecture Notes in Computer Science* 5594, pages 407–421, Brisbane, Australia, July 1–3, 2009. Springer, Heidelberg, Germany.

Wik18. D. Wikström. Verificatum. `https://github.com/verificatum/verificatum-vcr`, 2018.

YGW+21. Y. Yang, Z. Guan, Z. Wan, J. Weng, H. H. Pang, and R. H. Deng. Priscore: Blockchain-based self-tallying election system supporting score voting. *IEEE Transactions on Information Forensics and Security*, 16:4705–4720, 2021.

ZZH+25. Z. Zhou, Z. Zhang, F. Hao, B. Zheng, and Z. Masyhur. Qv-net: Decentralized self-tallying quadratic voting with maximal ballot secrecy. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, pages 453–467, New York, NY, USA, 2025. Association for Computing Machinery.