Statement Voting

Bingsheng Zhang[1] **and Hong-Sheng Zhou**[2]

1 **Lancaster University**
b.zhang2@lancaster.ac.uk
2 **Virginia Commonwealth University**
hszhou@vcu.edu

Abstract

The existing (election) voting systems, e.g., representative democracy, have many limitations and often fail to serve the best interest of the people in a collective decision-making process. To address this issue, the concept of liquid democracy has been emerging as an alternative decision-making model to make better use of "the wisdom of crowds". However, there is no known cryptographically secure e-voting implementation that supports liquid democracy.

In this work, we propose a new voting concept called *statement voting*, which can be viewed as a natural extension of the conventional voting approaches. In the statement voting, instead of defining a concrete election candidate, each voter can define a statement in his/her ballot but leave the vote "undefined" during the voting phase. During the tally phase, the (conditional) actions expressed in the statement will be carried out to determine the final vote. We initiate the study of statement voting under the Universal Composability (UC) framework, and propose several construction frameworks together with their instantiations. As an application, we show how statement voting can be used to realize a UC-secure liquid democracy voting system. We remark that our statement voting can be extended to enable more complex voting and generic ledger-based non-interactive multi-party computation. We believe that the statement voting concept opens a door for constructing a new class of e-voting schemes.

Keywords and phrases statement voting, liquid democracy, e-voting, universal composability

1 Introduction

Elections/Referendums provide people in each society with the opportunity to express their opinions in the collective decision making process. The existing election/voting systems can be mainly divided into two types, direct democracy and representative democracy. Unfortunately, either approach has many limitations, and it often fails to serve the best interest of the people. For example, to make correct decisions, the voters have to invest tremendous effort to analyze the issues. The cost of identifying the best voting strategy is high, even if we assume that the voter has collected accurate information. In addition, misinformation campaigns often influence the voters to select certain candidates which could be against the voters' true interests. We here ask the following challenging question:

> *Is it possible to introduce new technologies to circumvent the implementation barriers so that more effective democracy can be enabled?*

We very much expect an affirmative answer because from a societal perspective, we need to ensure that these unmotivated/misinformed voters to participate in the process of decision making.

A new concept. We could approach the above problem via multiple angles. In this paper, we propose a new and clean concept: *statement voting*. Statement voting can be viewed as a natural extension of traditional candidate voting. Instead of defining a fixed election candidate, each voter can define a statement in his or her ballot but leave the vote "undefined" during the voting phase. During the tally phase, the (conditional) actions expressed in the statement will be carried out to determine the final vote. Single Transferable Vote (STV) is a special case of statement voting, where the voters rank the election candidates instead of naming only one candidate in their ballots. The ranked candidate list together with the STV tally rule can be viewed as an outcome-dependent statement. Roughly speaking, the statement declares that if my favorite candidate has already won or has no chance to win, then I would like to vote for my second favorite candidate, and so on[1]. Liquid democracy [28] is another special case of statement voting; there, the voters can either vote directly on issues, or they can delegate their votes to representatives who vote on their behalf. The vote delegation can be expressed as a target-dependent statement, where a voter can define that his/her ballot is the same as the target voter's ballot. Of course, the target voter may also state whether he/she is willing to be delegated in the ballot.

Jumping ahead, in a statement voting, the ballot is in the form of $(\mathsf{ID}, \mathsf{targets}, \mathsf{statement})$, where ID is the voter's ID, $\mathsf{targets}$ is a set of target voters' IDs which will be referenced in the statement, and $\mathsf{statement}$ is the (conditional) statement. To realize liquid democracy voting, we can define the following simple statement: (i) if voter V_i wants to delegate his vote to V_j, then the ballot is $B := (\mathsf{V}_i, \{\mathsf{V}_j\}, \mathtt{delegate})$; (ii) if voter V_i wants to vote directly for election option opt, then the ballot is $B := (\mathsf{V}_i, \perp, \mathtt{vote}\ \mathsf{opt})$; and (iii) if the voter does not want to be delegated, then he can set his own ID to $\perp$. To obtain the basic intuition, let's first leave privacy aside and consider the following toy example.

Example: Take the Yes/No election as an example. Suppose there are seven ballots: $B_1 := (\mathsf{V}_1, \mathsf{V}_7, \mathtt{delegate})$, $B_2 := (\mathsf{V}_2, \perp, \mathtt{vote}\ \mathsf{Yes})$, $B_3 := (\mathsf{V}_3, \perp, \mathtt{vote}\ \mathsf{No})$, $B_4 := (\perp, \perp, \mathtt{vote}\ \mathsf{Yes})$, $B_5 := (\mathsf{V}_5, \mathsf{V}_4, \mathtt{delegate})$, $B_6 := (\perp, \mathsf{V}_3, \mathtt{delegate})$ and $B_7 := (\mathsf{V}_7, \mathsf{V}_3, \mathtt{delegate})$. Here, the effective vote of B_1 is defined by B_7, which is further defined by B_3; note that B_3 votes for No; that means, B_7 votes for No by following B_3. Now let's consider B_6: B_6 follows B_3; however, B_6 is not willing to be followed by anyone; as a result, B_6 also votes for No. Finally, let's consider B_5: B_5 follows B_4; however, B_4 is not willing to be followed by anyone; as a consequence, B_5 is re-defined as blank ballot, $\perp$. After interpreting the delegation statements, the final votes are $(\mathsf{No}, \mathsf{Yes}, \mathsf{No}, \mathsf{Yes}, \perp, \mathsf{No}, \mathsf{No})$.

Careful readers may wonder why this type of natural voting idea has never appeared in the *physical* world. Indeed, it is typically not available in the real life. Different from the toy example, in the reality, the voters care about privacy and anonymity. To ensure anonymity, the voters are not willing to leave their identities in the ballots. If no identities (or equivalences) are included in the ballots, then it is difficult for voters to "follow" other voters' choices. The election committees might assign each voter a temporal ID to achieve anonymity, but a voter needs to obtain the target voter's temporal ID in order to delegate his vote. This requires secure peer-to-peer channels among all the voters, which is not practical for a national election in the context of paper-voting.

Modeling statement voting. We provide a rigorous modeling for statement voting. More concretely, we model statement voting in the well-known Universal Composability (UC) framework, via an ideal functionality $\mathcal{F}_{\mathrm{SV}}$. The functionality interacts with a set of voters, trustees. In our formulation, we introduce a *family of functionalities* to facilitate various realizations. In practice, there is a trade-off between efficiency and privacy guarantees. More

[1] Note that this is not a complete description of STV. For those readers who are unfamiliar with STV, please see its full definition to avoid misunderstanding.

efficient constructs usually yield more privacy leakage. To fit various leakage scenarios, our ideal functional keep a working table $\mathbb{W}$ to trace the election transcripts. Depends on which parties are corrupted (and the scheme construction), some part of the working table will be leaked to the adversary.

Our toy example shows that it is possible to interpret the delegation statement by extending the conventional tally algorithm. However, it is not clear about how to apply the same technique in conjunction with privacy. At the beginning of the election, each voter can pick a temporal ID. However, the main challenge here is to distribute the temporal ID to the ones who need. The same as all existing end-to-end verifiable e-voting schemes, our design requires a publicly accessible consistent bulletin board, modeled as global functionality $\bar{\mathcal{G}}_{\mathrm{BB}}$. We let the voters post the re-randomizable threshold encryption of their temporal ID on the $\bar{\mathcal{G}}_{\mathrm{BB}}$. If voter V_i wants to delegate his ballot to voter V_j, he can include a re-randomized ciphertext of V_j's temporal ID. More specifically, V_i sets his ballot as $B_i := (w_i, W_j, \texttt{delegate})$, if he wants to delegate to V_j; or he sets $B_i := (w_i, \texttt{vote}, \mathsf{opt})$, if he wants to vote for opt; here w_i is V_i's temporal ID and W_j is the re-randomized ciphertexts of V_j's temporal ID. All the ballots will first be shuffled via a mix-net, and then all the trustees will jointly open those re-randomized ciphertexts inside the ballots. Subsequently, we can handle the delegation statement and compute the tally in the same way as the toy example.

Constructions. Our statement voting concept can be implemented via the following different approaches. We assume a trusted *Registration Authority* (RA) to ensure voter eligibility and a consistent *Bulletin Board* (BB) where the voting transactions and result will be announced to.

A fully/somewhat homomorphic encryption based scheme. In this scheme, the trustees first run a distributed key generation protocol to setup the voting public key $\textsc{pk}$. Each voter V_i then encrypt, sign and submit their *voting statements*, x_i (in forms of $(\mathsf{PID}_i, \mathsf{Enc}_{\textsc{pk}}(x_i))$) to the BB. To present re-play attacks, zero-knowledge (ZK) proofs are necessary to ensure the voter knows the plaintext included in his/her submitted ciphertext. After that, the tally processing circuit is evaluated over $\{(\mathsf{PID}_i, \mathsf{Enc}_{\textsc{pk}}(x_i))\}_{i\in[n]}$ by every trustee. The final tally ciphertext is then decrypted by the trustees and the result will be announced on the BB. More details can be found at Appendix B.

A verifiable MPC based scheme. In this scheme, we can adopt BDO publicly auditable MPC [6], where the trustees form the MPC system. They pre-compute sufficiently many correlated randomness (e.g., Beaver triples), and also set up a voting public key. Each voter V_i then encrypt, sign and submit their *voting statements*, x_i (in forms of $(\mathsf{PID}_i, \mathsf{Enc}_{\textsc{pk}}(x_i))$) to the BB. Again, to present re-play attacks, ZK proofs are necessary to ensure the voter knows the plaintext included in his/her submitted ciphertext. After that, the trustees perform MPC online computation to first decrypt those encrypted ballots and then evaluate the tally processing circuit over the shared ballots. Finally, the tally result will be posted on the BB. Note that during the online phase, BDO MPC scheme also posts audit information on the BB to enable public verifiability. More details can be found at Appendix C.

A mix-net based scheme. In this scheme, the trustees first run a distributed key generation protocol to set up the public key $\textsc{pk}$ of a re-randomizeable encryption scheme. Each voter V_i then encrypt, sign and submit a random temporal ID w_i, in forms of $(\mathsf{PID}_i, \mathsf{Enc}_{\textsc{pk}}(w_i))$ to the BB. After that each voter will submit an encrypted voting statement where PID_j are replaced with re-randomized encryption τ_j, for all $j \in [n]$. The encrypted statement together with the voter's encrypted temporal ID will then be shuffled via a mix-net. The resulting ciphertexts will be decrypted by the trustees and evaluated by every voter themselves. Note that the tally processing function must be symmetric, otherwise we cannot use mix-net. More details can be found at Section 3.

An immediate application: Liquid democracy. In the past decades, the concept of liquid democracy [28] has been emerging as an alternative decision making model to make better use of collective intelligence. Liquid democracy is a hybrid of direct democracy and representative democracy, where the voters can either vote directly on issues, or they can delegate their votes to representatives who vote on their behalf. Due to its advantages, liquid democracy has received high attentions since the spread of its concept; however, there is no satisfactory solution in the form of either paper-voting or e-voting yet. We show how to achieve liquid democracy as an application of statement voting.

Extensions and further remarks. In this work, we initiate the study of statement voting and liquid democracy. We remark that our statement voting concept can be significantly extended to support much richer ballot statements. It opens a door for constructing a new class of e-voting schemes. We also note that this area of research is far from being completed, and our design and modeling ideas can be further improved. For example, if there is a delegation loop in which a set of voters delegate their votes to each other while no one votes, then what should be the "right" policy? Should the ballots be reset as blank ballots? This might not be ideal in reality. One possible approach is to extend the delegation statement to include a default vote. When a delegation loop exists, the involved ballots could be counted as their default votes. We finally emphasize that, voting policies can be heavily influenced by local legal and societal conditions. How to define "right" voting policy itself is a very interesting question. We believe our techniques here have the potential to help people to identify suitable voting policies which can further eliminate the barriers to democracy. See Sec. 5 for further discussion.

Related work. The concept of liquid democracy (a.k.a. delegative democracy) is emerging over the last decades [39, 2, 11]. To our best knowledge, Ford [28] first officially summarized the main characteristics of liquid democracy and brought it to the vision of computer science community. However, in terms of implementation/prototyping, there was no system that can enable liquid democracy until very recently. All the existing liquid democracy voting systems only focus on the functionality aspect of liquid democracy, and no privacy or some other advanced security properties were considered. For instance, Google Votes [33] is a decision-making system that can support liquid democracy, and it is built on top of social networks, e.g., the internal corporate Google+ network. In terms of UC modeling on e-voting. Groth [30] gave the first UC definition for an e-voting system, and he proposed a protocol using (threshold) homomorphic encryption. Moran and Naor [40] later studied the privacy and receipt-freeness of an e-voting system in the stand-alone setting. Unruh and Muller-Quade [47] gave a formal study of e-voting coerciability in the UC framework. Alwen *et al.* [4] considered stronger versions of coerciability in the MPC setting under UC framework. Almost all the end-to-end verifiable e-voting systems [1, 25, 23, 35, 34] requires a consistent bulletin board (BB). However, none of them gives a practical realization of BB.

2 Modeling

The parties involved in a statement voting system are a set of trustees $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$, and a set of voters $\mathbb{V} := \big\{\mathsf{V}_1, \ldots, \mathsf{V}_n\big\}$. In this section, we will define an ideal functionality for statement voting. Preliminary can be found in Appendix A. In Appendix B, Appendix C and Sec. 3, we will construct several protocols for realizing the ideal statement voting functionality.

Functionality $\mathcal{F}_{\text{SV}}$

The functionality $\mathcal{F}_{\text{SV}}$ interacts with voters $\mathbb{V}$, trustees $\mathbb{T}$, and the adversary $\mathcal{S}$. It is parameterized by an algorithm $\mathsf{TallyProcess}$ (see Fig. 2), a working table $\mathbb{W}$, and variables *result*, T_1, T_2, and B_i for all $i \in [n]$. Let $\mathbb{V}_{\mathsf{honest}}$, $\mathbb{V}_{\mathsf{corrupt}}$ and $\mathbb{T}_{\mathsf{honest}}$, $\mathbb{T}_{\mathsf{corrupt}}$ denote the set of honest/corrupt voters and trustees, respectively.
Initially, set $\mathit{result} := \emptyset$, $T_1 := \emptyset$, $T_2 := \emptyset$; for $i \in [n]$, set $B_i := \emptyset$.
Table $\mathbb{W}$ consists of n entries, and each entry consists of voter's real ID, voter's alternative ID, and the statement that the voter submitted; for all $i \in [n]$, the ith entry $\mathbb{W}[i] := (\mathsf{V}_i, w_i, \mathit{statement}_i)$, where $w_i \leftarrow \{0,1\}^\lambda$, $\mathit{statement}_i := \emptyset$.

Preparation:

1. Upon receiving input $(\textsc{InitialTrustee}, \mathsf{sid})$ from the trustee $\mathsf{T}_j \in \mathbb{T}$, set $T_1 := T_1 \cup \{\mathsf{T}_j\}$, and send $(\textsc{InitialTrusteeNotify}, \mathsf{sid}, \mathsf{T}_j)$ to $\mathcal{S}$.

Ballot Casting:

1. Upon receiving input $(\textsc{Cast}, \mathsf{sid}, (s_i, w_i^*))$ from $\mathsf{V}_i \in \mathbb{V}$, if $|T_1| < k$, ignore it. Otherwise,
   - if V_i is honest ($w_i^* := \perp$), update $\mathbb{W}[i] := (\mathsf{V}_i, w_i, s_i)$; send $(\textsc{CastNotify}, \mathsf{sid}, \mathsf{V}_i)$ to $\mathcal{S}$.
   - if V_i is corrupt, then update $\mathbb{W}[i] := (\mathsf{V}_i, w_i^*, s_i)$.

   If $|\mathbb{T}_{\mathsf{corrupt}}| = k$, then additionally send a message $(\textsc{Leak}, \mathsf{sid}, \mathbb{W}[i])$ to $\mathcal{S}$.

Tally:

1. Upon receiving input $(\textsc{Tally}, \mathsf{sid})$ from the trustee $\mathsf{T}_j \in \mathbb{T}$, set $T_2 := T_2 \cup \{\mathsf{T}_j\}$ and
   - set $\mathbb{U} := \mathbb{W}$; then eliminate all V_i's in $\mathbb{U}$; sort the entries in $\mathbb{U}$ lexicographically.
   - define L. For example, set $L := \mathsf{TallyProcess}(\mathbb{U})$ or $L := \mathbb{U}$ or $L := \mathbb{W}$.

   Send a notification message $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_j)$ to $\mathcal{S}$.
   If $|T_2 \cap \mathbb{T}_{\mathsf{honest}}| + |\mathbb{T}_{\mathsf{corrupt}}| = k$, send a leakage message $(\textsc{Leak}, \mathsf{sid}, L)$ to $\mathcal{S}$.
   If $|T_2| = k$, compute $\mathit{result} \leftarrow \mathsf{TallyProcess}(\mathbb{U})$.
2. Upon receiving input $(\textsc{ReadResult}, \mathsf{sid})$ from a voter $\mathsf{V}_i \in \mathbb{V}$, if $\mathit{result} = \emptyset$, ignore the input. Else, return $(\textsc{ResultReturn}, \mathsf{sid}, \mathit{result})$ to V_i.

Figure 1 The voting functionality $\mathcal{F}_{\text{SV}}$.

The statement voting functionality. The ideal functionality for statement voting, denoted as $\mathcal{F}_{\text{SV}}$, is formally described in Fig. 1. The functionality interacts with n number of voters, k number of trustees. Let $\mathbb{V}_{\mathsf{honest}}$, $\mathbb{V}_{\mathsf{corrupt}}$ and $\mathbb{T}_{\mathsf{honest}}$, $\mathbb{T}_{\mathsf{corrupt}}$ denote the set of honest/corrupt voters and trustees, respectively. It consists of three phases—Preparation, Ballot Casting, and Tally. The functionality uses a working table $\mathbb{W}$ to trace the voters' behavior during the entire ideal execution. Each entry of the working table is saved for storing one voter's information including the voter's original ID, his alternative ID, and the voting statement that he submitted;

Preparation phase. During the preparation phase, the trustees, playing the role of voting organizers, need to indicate their presence to $\mathcal{F}_{\text{SV}}$ by sending $(\textsc{InitialTrustee}, \mathsf{sid})$ to it. The election/voting will not start until all the trustees have participated in the preparation.

Ballot Casting phase. During the ballot casting phase, each voter can submit his voting statement, and this voting statement will be recorded in the corresponding entry. If a voter is corrupt, then he is also allowed to revise his own alternative ID in the working table. More concretely, based on the input $(\textsc{Cast}, \mathsf{sid}, (s_i, w_i^*))$ from voter V_i, the corresponding entry will be updated, i.e., $\mathbb{W}[i] := (\mathsf{V}_i, w_i, s_i)$ if the voter is honest, and $\mathbb{W}[i] := (\mathsf{V}_i, w_i^*, s_i)$ if V_i is corrupt. When all the trustees are corrupted, the functionality $\mathcal{F}_{\text{SV}}$ leaks the entire working tape of the election transcript (i.e., $\mathbb{W}$), to the adversary.

Tally phase. Voters' information in the working table $\mathbb{W}$ will be used in the tally phase for defining the privacy leakage as well as the final result. More concretely, we compute a new table $\mathbb{U}$ by first eliminating all V_i's in $\mathbb{W}$, and then sorting all the entries lexicographically. This carefully defined table $\mathbb{U}$ can now be used to define (1) the final result via applying a circuit $\mathsf{TallyProcess}$ on $\mathbb{U}$, and (2) certain level of privacy leakage L. Our formulation here allows us to define a *class* of statement voting functionalities. For example, to define a functionality with strong privacy, we can set $L := \mathsf{TallyProcess}(\mathbb{U})$; we can also set $L := \mathbb{U}$ to define a functionality with relatively weaker privacy, or set $L := \mathbb{W}$ to define a functionality without privacy.

TallyProcess

Input: a set of ballots $\mathcal{B} := (B_1, \ldots, B_n)$
Output: the tally result *result*

Statement interpretation:

- Compute $(v_1, \ldots, v_n) \leftarrow \mathsf{StatementProcess}(B_1, \ldots, B_n)$, where StatementProcess takes input as the set of statements and outputs the voters' final votes.

Tally computation:

- Compute $result \leftarrow \mathsf{TallyAlg}(v_1, \ldots, v_n)$, where $\mathsf{TallyAlg}(\cdot)$ is the tally algorithm that takes input as the votes and outputs the tally result.
- Return *result*.

Figure 2 The extended tally processing algorithm.

3 Mix-net based construction

In this section, we present an efficient statement voting construction based on mix-net. The privacy that this construction achieves is known as pseudonymity. He emphasize that this level of privacy has been widely accepted and is consistent with all the existing paper-based voting systems. In Appendix B and Appendix C, we also propose fully/somewhat homomorphic encryption (FHE) based scheme and a publicly verifiable MPC based scheme (where the trustees jointly perform MPC computation of the tally circuit in the server-client model). As mentioned before, there is a trade-off between efficiency and privacy guarantee. While FHE and MPC based solutions offer better privacy, they are generally less efficient.

Before formally describing our mix-net based scheme, we first provide an intuition. At the beginning of each election, the voters V_i, $i \in [n]$, are assigned with a temporal random ID, denoted as ID_i. Let $\mathcal{I} := \{\mathsf{ID}_1, \ldots, \mathsf{ID}_n\}$ be the set of all the voter's random IDs. The voter's statement takes input as a subset of $\mathcal{I}$, denoted as $\mathcal{D}$, and uses $\mathsf{ID} \in \mathcal{D}$ as references to point to those voters' ballots that will be involved in the statement execution. For instance, the statement could be "*If both voter ID_x and voter ID_y vote for 'Yes', then my vote is 'Yes'; otherwise, my vote is 'No'.*" The ballot of a voter V_i is in forms of $B_i := (\mathsf{ID}_i, \mathsf{statement}_i(\mathcal{D}))$, where ID_i is the voter's temporal ID, and $\mathsf{statement}_i$ is the voter's statement that takes $\mathcal{D}$ as a parameter. To ensure privacy, the voters cannot post their temporal IDs publicly on the bulletin board $\bar{\mathcal{G}}_{\mathrm{BB}}$; however, the voters should be allowed to freely refer to any voter's ID.

To address this challenge, we introduce the following technique. Before the ballot casting phase, each voter picks a random ID and posts the (re-randomizable) encryption of the ID on the $\bar{\mathcal{G}}_{\mathrm{BB}}$. If a voter wants to refer to another voter in the statement, he/she simply copies the ciphertext of the corresponding voter's ID. At the tally phase, all the ballots are passing through re-encryption based mix-net, and then are decrypted to calculate the statements and tally result. We remark that in practice the mix-net servers can be different from talliers (a.k.a. decrypters). As such, they could have different threshold requirements. For notation simplicity, we combine both roles to the same set of parties, trustees, in the protocol description.

3.1 Protocol description

In this section, we formally describe our mix-net based construction for statement voting. The protocol is designed in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world and it consists of three phases: preparation, ballot casting, and tally. For the sake of notation simplicity, we omit the processes of filtering invalid messages on $\bar{\mathcal{G}}_{\mathrm{BB}}$. In practice, $\bar{\mathcal{G}}_{\mathrm{BB}}$ contains many messages with invalid signatures, and all those messages should be ignored. We will use threshold re-randomizable encryption (TRE) as a building block. A threshold re-randomizable encryption scheme TRE consists of a tuple of algorithms: $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CombinePK}, \mathsf{CombineSK}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{ReRand})$. More details can be found in Appendix D.2.

Preparation phase. As depicted in Fig. 3, in the preparation phase, each trustee T_j, $j \in [k]$ first picks a randomness generates α_j and generates a partial public key using $(\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) \leftarrow \mathsf{TRE.Keygen}(\mathsf{param}; \alpha_j)$. It then generates an NIZK proof

$$\pi_j^{(1)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_4}\left\{ \ (\overline{\mathtt{pk}}_j), (\alpha_j, \overline{\mathtt{sk}}_j) : (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{TRE.Keygen}(\mathsf{param}; \alpha_j) \ \right\}$$

to show that this process is executed correctly; namely, it shows knowledge of $(\alpha_j, \overline{\mathtt{sk}}_j)$ w.r.t. to the generated partial public key $\overline{\mathtt{pk}}_j$. It then signs and posts $(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Preparation

Upon receiving $(\textsc{InitialTrustee}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the trustee T_j, $j \in [k]$, operates as the follows:

- Generate $(\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) \leftarrow \mathsf{TRE.Keygen}(\mathsf{param}; \alpha_j)$ where α_j is the fresh randomness, and then compute
$$\pi_j^{(1)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_4}\left\{ \ (\overline{\mathtt{pk}}_j), (\alpha_j, \overline{\mathtt{sk}}_j) : (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{TRE.Keygen}(\mathsf{param}; \alpha_j) \ \right\}$$
- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and receives $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)} \rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Figure 3 Mix-net based statement voting scheme $\Pi_{\textsc{mix-sv}}$ in $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part I).

Ballot casting phase. As depicted in Fig. 4, the ballot casting phase consists of two rounds. In the first round, each voter V_i, $i \in [n]$ first fetches the trustees' partial public keys $\{\overline{\mathtt{pk}}_j\}_{j=1}^k$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. She then checks the validity of their attached NIZK proofs. If all the NIZK proofs are verified, she computes and stores the election public key as $\mathtt{pk} \leftarrow \mathsf{TRE.CombinePK}(\{\overline{\mathtt{pk}}_j\}_{j=1}^k)$. In addition, the voter V_i picks a random temporal ID $w_i \leftarrow \{0,1\}^\lambda$. She then uses the election public key $\mathtt{pk}$ to encrypt w_i as $W_i \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, w_i; \beta_i)$ with fresh randomness β_i. She also computes the corresponding NIZK

$$\pi_i^{(2)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_5}\{(\mathtt{pk}, W_i), (\beta_i, w_i) : W_i = \mathsf{TRE.Enc}(\mathtt{pk}, w_i; \beta_i)\}$$

to show she is the creator of this ciphertext. Voter V_i then signs and posts $(W_i, \pi_i^{(2)})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

In the second round, each voter V_i, $i \in [n]$ first fetches all the posted encrypted temporal IDs from $\bar{\mathcal{G}}_{\mathrm{BB}}$, and checks their attached NIZK proofs. For any missing or invalid (encrypted)

Ballot Casting

Upon receiving $(\textsc{Cast}, \mathsf{sid}, s_i)$ from the environment $\mathcal{Z}$, the voter V_i operates as the follows:

∘ Round 1:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. If $\left\{\langle \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)} \rangle\right\}_{j\in[k]}$ is contained in state, then for $j \in [k]$, send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), b_j^{(1)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; If $\prod_{j=1}^{k} b_j^{(1)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_4}.\mathsf{Verify}(\overline{\mathtt{pk}}_j, \pi_j^{(1)}) = 1$ for $j \in [k]$. If any of the checks is invalid, halt.
- Compute and store $\mathtt{pk} \leftarrow \mathsf{TRE.CombinePK}(\{\overline{\mathtt{pk}}_j\}_{j=1}^{k})$.
- Randomly selects $w_i \leftarrow \{0,1\}^{\lambda}$ and compute $W_i \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, w_i; \beta_i)$ with fresh randomness β_i together with
$$\pi_i^{(2)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_5}\left\{ \ (\mathtt{pk}, W_i), (\beta_i, w_i) : W_i = \mathsf{TRE.Enc}(\mathtt{pk}, w_i; \beta_i) \ \right\} .$$
- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (W_i, \pi_i^{(2)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (W_i, \pi_i^{(2)}), \sigma_i^{(2)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{V}_i, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (W_i, \pi_i^{(2)}), \sigma_i^{(2)} \rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

∘ Round 2:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. For $\ell \in [n]$, if $\langle \mathsf{ssid}, (W_\ell, \pi_\ell^{(2)}), \sigma_\ell^{(2)} \rangle$ is contained in state, then send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (W_\ell, \pi_\ell^{(2)}), \sigma_\ell^{(2)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (W_\ell, \pi_\ell^{(2)}), b_j^{(2)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; For $\ell \in [n]$, set $W_\ell \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, \perp; 0)$ if W_ℓ is missing or $b_\ell^{(2)} = 0$ or $\mathsf{NIZK}_{\mathcal{R}_5}.\mathsf{Verify}((\mathtt{pk}, W_\ell), \pi_\ell^{(2)}) = 0$.
- Set $\ell := 1$. Scan through the statement s_i, for each referenced voter V_j, compute
  - $U_{i,\ell} \leftarrow \mathsf{TRE.ReRand}(\mathtt{pk}, W_j; \gamma_{i,\ell})$ with a fresh randomness $\gamma_{i,\ell}$ and
$$\pi_{i,\ell}^{(3)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_6}\left\{ \begin{array}{c} (\mathtt{pk}, (W_0, \ldots, W_n), U_{i,\ell}), (\gamma_{i,\ell}, j) : \\ U_{i,\ell} = \mathsf{TRE.ReRand}(\mathtt{pk}, W_j; \gamma_{i,\ell}) \end{array} \right\}$$
  - Replace V_j with label 'U_ℓ' in the statement s_i. Set $\ell := \ell + 1$ and repeat the above process for all the voter IDs in s_i.
  - If $\ell < \lambda_1$, compute
    - $U_{i,\ell} \leftarrow \mathsf{TRE.ReRand}(\mathtt{pk}, W_0; \gamma_{i,\ell})$ with a fresh randomness $\gamma_{i,\ell}$ and
$$\pi_{i,\ell}^{(3)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_6}\left\{ \begin{array}{c} (\mathtt{pk}, (W_0, \ldots, W_n), U_{i,\ell}), (\gamma_{i,\ell}, 0) : \\ U_{i,\ell} = \mathsf{TRE.ReRand}(\mathtt{pk}, W_0; \gamma_{i,\ell}) \end{array} \right\}$$
    - Repeat the above process until $\ell = \lambda_1$.
  - Denote the modified statement as s'_i. Compute $S_i \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, s'_i; \delta_i)$ and $\pi_i^{(4)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_5}\left\{ \ ((\mathtt{pk}, S_i), (\delta_i, s'_i) : S_i = \mathsf{TRE.Enc}(\mathtt{pk}, s'_i; \delta_i) \ \right\}$.
- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, ((U_{i,\ell}, \pi_{i,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_i, \pi_i^{(4)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{V}_i, \mathsf{ssid}')$ for some ssid', and receive $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, ((U_{i,\ell}, \pi_{i,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_i, \pi_i^{(4)}), \sigma_i^{(3)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, ((U_{i,\ell}, \pi_{i,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_i, \pi_i^{(4)}), \sigma_i^{(3)} \rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Figure 4 Mix-net based statement voting scheme $\Pi_{\text{MIX-SV}}$ in $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part II).

temporal IDs, the voters replace them with $\mathsf{TRE.Enc}(\mathtt{pk}, \perp; 0)$, which is the encryption of $\perp$ with trivial randomness. Moreover, the voters also defines $W_0 \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, \perp; 0)$.

For the sake of uniformity, our scheme restricts that each voter's statement can refer up to $\lambda_1 \in \mathbb{N}$ the other voters' IDs, and the size of the statement should fit in the plaintext space. For a voter V_i, $i \in [n]$, denote $\mathcal{D}_i \subseteq [n]$ as the set of indices of the referenced voters' IDs. Let $\mathcal{W}_i := \{W_j \mid j \in \mathcal{D}_i\}$ be the set of ciphertexts of the corresponding referenced voters' IDs.

The voter V_i re-randomizes all the ciphertexts in $\mathcal{W}_i$ and pads re-randomized W_0's to form a ciphertext vector of size λ_1, denoted as $(U_1, \ldots, U_{\lambda_1})$. The voter V_i then replaces the voter IDs in the statement as the pointers to U_j, $j \in [\lambda_1]$. Note that to avoid nested ciphertext, the statement uses label 'U_j' instead of the actual ciphertext U_j. Denote the modified statement as s'_i. It then encrypts s'_i to ciphertext S_i. Of course, to ensure correctness, NIZK proofs are generated to show (i) U_j, $j \in [\lambda_1]$ is indeed re-randomized from one of the ciphertexts in $(W_0, \ldots, W_n)$, and (ii) S_i is indeed created by the voter himself/herself. The voter V_i then signs and posts $(U_1, \ldots, U_{\lambda_1})$ and S_i together with the corresponding NIZK proofs to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Tally (Part I)

Upon receiving $(\textsc{Tally}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the trustee T_j, where $j \in [k]$, operates as the follows:

∘ Round 1 to k:

- If $j = 1$, send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. For $x \in [n]$:
  - If $\langle \mathsf{ssid}, (W_x, \pi_x^{(2)}), \sigma_x^{(2)} \rangle$ is contained in *state*, then send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (W_x, \pi_x^{(2)}), \sigma_x^{(2)})$ to $\widehat{\mathcal{F}}_{\mathrm{CERT}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (W_x, \pi_x^{(2)}), b_x^{(2)})$ from $\widehat{\mathcal{F}}_{\mathrm{CERT}}$;
  - If $\langle \mathsf{ssid}, ((U_{x,\ell}, \pi_{x,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_x, \pi_x^{(4)}), \sigma_x^{(3)} \rangle$, is contained in *state*, then send
  $$(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, ((U_{x,\ell}, \pi_{x,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_x, \pi_x^{(4)}), \sigma_x^{(3)})$$
  to $\widehat{\mathcal{F}}_{\mathrm{CERT}}$, receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, ((U_{x,\ell}, \pi_{x,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_x, \pi_x^{(4)}), b_x^{(3)})$ from $\widehat{\mathcal{F}}_{\mathrm{CERT}}$;

  Set $i := 0$. For $\ell \in [n]$, define $e_i^{(0)} := (W_x, (U_{x,\ell})_{\ell=1}^{\lambda_1}, S_x)$ and $i := i + 1$ if the following holds:
  - $W_x, (U_{x,\ell})_{\ell=1}^{\lambda_1}, S_x$ exist in *state* and $b_x^{(2)} \cdot b_x^{(3)} = 1$;
  - $\mathsf{NIZK}_{\mathcal{R}_5}.\mathsf{Verify}((\mathtt{pk}, W_x), \pi_x^{(2)}) = 1$;
  - For all $\ell \in [\lambda_1]$, $\mathsf{NIZK}_{\mathcal{R}_6}.\mathsf{Verify}((\mathtt{pk}, (W_0, \ldots, W_n), U_{x,\ell}), \pi_{x,\ell}^{(3)}) = 1$;
  - $\mathsf{NIZK}_{\mathcal{R}_5}.\mathsf{Verify}((\mathtt{pk}, S_x), \pi_x^{(4)}) = 1$;

  (Set $n' := i$ after the above process.)
- (If $j > 1$, T_j sends $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$; T_j then fetches $(e_i^{(j-2)})_{i=1}^{n'}, e_i^{(j-1)})_{i=1}^{n'}, \pi_{j-1}^{(5)}$ from *state* and check $\mathsf{NIZK}_{\mathcal{R}_7}.\mathsf{Verify}((\mathtt{pk}, (e_1^{(j-2)}, \ldots, e_{n'}^{(j-2)}), (e_1^{(j-1)}, \ldots, e_{n'}^{(j-1)})), \pi_{j-1}^{(5)}).)$
  T_j randomly picks a permutation Π_j over [n']; For $i \in [n']$, for $\ell \in [\lambda_2]$: set $e_{i,\ell}^{(j)} \leftarrow \mathsf{TRE.ReRand}(\mathtt{pk}, e_{\Pi_j(i),\ell}^{(j-1)}; r_{i,\ell}^{(j)})$, where $r_{i,\ell}^{(j)}$ are fresh randomness. Compute
  $$\pi_j^{(5)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_7} \left\{ \begin{array}{c} \left(\mathtt{pk}, (e_1^{(j-1)}, \ldots, e_{n'}^{(j-1)}), (e_1^{(j)}, \ldots, e_{n'}^{(j)})\right), \left(\Pi_j, (r_{i,\ell}^{(j)})_{i\in[n'],\ell\in[\lambda_1+2]}\right) \; : \\ \forall i \in [n'] \; \forall \ell \in [\lambda_1 + 2] \; : \quad e_{i,\ell}^{(j)} = \mathsf{TRE.ReRand}\left(\mathtt{pk}, e_{\Pi_j(i),1}^{(j-1)} \; ; r_{i,\ell}^{(j)}\right) \end{array} \right\}$$
- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}))$ to $\widehat{\mathcal{F}}_{\mathrm{CERT}}$ and receive $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), \sigma_j^{(4)})$ from $\widehat{\mathcal{F}}_{\mathrm{CERT}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}, \sigma_j^{(4)} \rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Figure 5 Mix-net based statement voting scheme $\Pi_{\text{MIX-SV}}$ in $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\mathrm{CERT}}\}$-hybrid world (Part III).

Tally phase. The tally phase is depicted in Fig. 5 and Fig. 6. The trustees first fetches $(W_i, (U_1, \ldots, U_{\lambda_1}), S_i)$ (which is viewed as the submitted ballot for voter V_i) from $\bar{\mathcal{G}}_{\mathrm{BB}}$ and check their attached NIZK proofs. All the invalid ballots will be discard. Let n' be the

Tally (Part II)

∘ Round $k+1$:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \textit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. For $j \in [k]$, if $\langle \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}, \sigma_j^{(4)} \rangle$ is contained in *state*, then send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), \sigma_j^{(4)})$ to $\widehat{\mathcal{F}}_{\mathrm{CERT}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), b_j^{(4)})$ from $\widehat{\mathcal{F}}_{\mathrm{CERT}}$; if $b_j^{(4)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_4}.\mathsf{Verify}((\mathtt{pk}, (e_1^{(j-1)}, \ldots, e_{n'}^{(j-1)}), (e_1^{(j)}, \ldots, e_{n'}^{(j)})), \pi_j^{(5)}) = 1$. If any of the above checks is invalid, halt.
- For $i \in [n']$, $\ell \in [\lambda_1 + 2]$, compute $\overline{m}_{i,\ell}^{(j)} \leftarrow \mathsf{TRE.ShareDec}(\overline{\mathtt{sk}}_j, e_{i,\ell}^{(k)})$ and

$$\pi_{i,j,\ell}^{(6)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_8} \left\{ \begin{array}{c} (e_{i,\ell}^{(k)}, \overline{m}_{i,\ell}^{(j)}, \overline{\mathtt{pk}}_j), (\overline{\mathtt{sk}}_j, \alpha_j) : \\ (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{TRE.Keygen}(\mathsf{param}; \alpha_j) \\ \wedge\, \overline{m}_{i,\ell}^{(j)} = \mathsf{TRE.ShareDec}(\overline{\mathtt{sk}}_j, e_{i,\ell}^{(k)}) \end{array} \right\}$$

- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,\ell}^{(j)}, \pi_{i,j,\ell}^{(6)})_{i\in[n'],\ell\in[\lambda_1+2]})$ to $\widehat{\mathcal{F}}_{\mathrm{CERT}}$ and receives $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,\ell}^{(j)}, \pi_{i,j,\ell}^{(6)})_{i\in[n'],\ell\in[\lambda_1+2]}, \sigma_j^{(5)})$ from $\widehat{\mathcal{F}}_{\mathrm{CERT}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (\overline{m}_{i,\ell}^{(j)}, \pi_{i,j,\ell}^{(6)})_{i\in[n'],\ell\in[\lambda_1+2]}, \sigma_j^{(5)} \rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Upon receiving $(\textsc{ReadResult}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the voter V_i, where $i \in [n]$, operates as the follows:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \textit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$.
  For $j \in [k]$, if $\langle \mathsf{ssid}, (\overline{m}_{i,\ell}^{(j)}, \pi_{i,j,\ell}^{(6)})_{i\in[n'],\ell\in[\lambda_1+2]}, \sigma_j^{(5)} \rangle$ is contained in *state*, send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,\ell}^{(j)}, \pi_{i,j,\ell}^{(6)})_{i\in[n'],\ell\in[\lambda_1+2]})_{i\in[n'],\ell\in[\lambda_1+2]}, \sigma_j^{(5)})$ to $\widehat{\mathcal{F}}_{\mathrm{CERT}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,\ell}^{(j)}, \pi_{i,j,\ell}^{(6)})_{i\in[n'],\ell\in[\lambda_1+2]}, b_j^{(5)})$ from $\widehat{\mathcal{F}}_{\mathrm{CERT}}$. If $\prod_{j=1}^{k} b_j^{(5)} = 1$, for all $j \in [k], i \in [n'], \ell \in [\lambda_1 + 2]$, check $\mathsf{NIZK}_{\mathcal{R}_8}.\mathsf{Verify}((e_{i,\ell}^{(k)}, \overline{m}_{i,\ell}^{(j)}, \overline{\mathtt{pk}}_i), \pi_{i,j,\ell}^{(6)}) = 1$. If any of the above checks is invalid, return $(\textsc{Error}, \mathsf{sid})$ to the environment $\mathcal{Z}$ and halt.
- For $i \in [n']$, $\ell \in [\lambda_1 + 2]$: compute $m_{i,\ell} \leftarrow \mathsf{TRE.ShareCombine}(e_{i,\ell}^{(k)}, \{\overline{m}_{i,\ell}^{(j)}\}_{j=1}^{k})$, $\ell \in [\lambda_1 + 2]$; define $B_i := (m_{i,\ell})_{\ell\in[\lambda_1+2]}$.
- Calculate election result *result* $\leftarrow \mathsf{TallyProcess}(\{B_i\}_{i\in[n']})$, and return $(\textsc{ReadResultReturn}, \mathsf{sid}, \textit{result})$ to $\mathcal{Z}$.

Figure 6 Mix-net based statement voting scheme $\Pi_{\text{MIX-SV}}$ in $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\mathrm{CERT}}\}$-hybrid world (Part IV).

number of valid ballots. All the trustees then jointly shuffle the ballots via a re-encryption mix-net. More specifically, each trustee sequentially permutes $(W_i, (U_1, \ldots, U_{\lambda_1}), S_i)$ as a bundle using shuffle re-encryption. To ensure correctness, the trustee also produces a NIZK proof showing the correctness of the shuffle re-encryption process. After that, upon receiving $(\textsc{Tally}, \mathsf{sid})$ from the environment, all the trustees T_j check the correctness of the entire mix-net and then jointly decrypt the mixed ballots using $\mathsf{TRE.ShareDec}$. More specifically, each trustee will sign and post its decryption shares to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Each voter can then compute the tally result as follows. The voter first fetches all the decryption shares and checks their validity using $\mathsf{NIZK}_{\mathcal{R}_8}.\mathsf{Verify}$. Upon success, the voter uses $\mathsf{TRE.ShareCombine}$ to reconstruct the messages. She then use $\mathsf{TallyProcess}$ as described in Fig. 2 to calculate the final tally.

▶ Remark. The re-randmonizable encryption (TRE) scheme used in this scheme can be replaced by a re-randomizable RCCA encryption scheme. Here RCCA is the short name for *replayable CCA* defined by Canetti, Krawczyk, and Nielsen [19]. There are a few RCCA constructions [31, 44, 22, 21] in the literature. In our scheme, it is possible to distribute a publicly verifiable RCCA encryption scheme, e.g. [22] and use it as an enhanced TRE. Subsequently, $\mathsf{NIZK}_{\mathcal{R}_6}$ can be removed. Since the running time of proving and verifying $\mathsf{NIZK}_{\mathcal{R}_6}$ is linear to the number of voters n, it is more efficient to use RCCA instead of TRE for large n in practice.

3.2 Security and Instantiation

In Appendix D.3, we show how to instantiate the TRE, and in Appendix D.4, we show how to instantiate all the associated NIZK proofs. We prove the security of the mix-net based scheme with the following the theorem.

▶ **Theorem 1.** *Protocol $\Pi_{\text{MIX-SV}}$ described in Figure 3, Figure 4, Figure 5 and Figure 6 UC-realizes $\mathcal{F}_{\text{SV}}$ in the $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}\}$-hybrid world against static corruption.*

The security proof can be found in Appendix D.1.

4 Application to Liquid Democracy

As mentioned before, *liquid democracy* is an emerging type of voting system that receives high attentions since the spread of its concept; however, there is no satisfactory solution in the form of either paper-voting or e-voting yet[2]. We now show that how to define a simple statement to enable liquid democracy. We are particularly interested in the mix-net based scheme due to its efficiency. In the following, we will realize a liquid democracy voting scheme on top of the scheme presented in Section 3.

The preparation phase of the liquid democracy scheme is identical to Fig. 3 In the ballot casting phase, in round 1, the voter V_i, $i \in [n]$ picks a random temporal ID, and submits its encryption to $\bar{\mathcal{G}}_{\text{BB}}$ as described in Fig. 4. In round 2, the liquid democracy statement consists of two ciphertexts (U, S); if the voter V_i wants to delegate her vote to voter V_j, she sets $U \leftarrow \mathsf{TRE.ReRand}(\mathtt{pk}, W_j)$ and $S \leftarrow \mathsf{TRE.Enc}(\perp)$; if the voter V_i wants to directly cast her vote x_i, she sets $U \leftarrow \mathsf{TRE.ReRand}(\mathtt{pk}, W_0)$ and $S \leftarrow \mathsf{TRE.Enc}(x_i)$. The tally phase is also identical to the one depicted in Fig. 5 and Fig. 6.

The statement interpretation step in the $\mathsf{TallyProcess}$ is defined as follows. Each ballots is in form of either $B_i = (w_i, u_i, \perp)$ or $B_i = (w_i, \perp, x_i)$, where w_i and u_i are temporal ID's, and x_i is a vote. To resolve the delegation, the algorithm needs to follow the "chain of delegation", i.e., for each ballot B_i:

- If B_i is in form of $(w_i, u_i, \perp)$, try to locate a ballot B_j in form of (u_i, X, Y). If founded, replace $B_i := (w_i, X, Y)$.
- Repeat the above step, until B_i is in form of $(w_i, \perp, Z)$. If there is a delegation loop, define $B_i := (w_i, \perp, \perp)$.

In case of delegation loop, we set the ballot to blank ballot. Of course, we can enrich the statement by adding another variable to indicate whether a voter wants to be delegated.

[2] All the existing liquid democracy implementations do not consider privacy/anonymity. This drawback prevents them from being used in serious elections. Here, we note that straightforward blockchain-based solutions cannot provide good privacy in practice. Although some blockchains (e.g., Zerocash [8]) can be viewed as a global mixer, they implicitly require anonymous channels. In practice, all the implementations of anonymous channels suffer from time leakage, i.e., the user's ID is only hidden among the other users who are also using the system at the same time. Subsequently, the adversary may easily identify the user during quiet hours.

When the “chain of delegation" breaks by V_i wants to delegate his vote to V_j, while V_j does not want to be delegated. In this case, V_i's ballot will be re-set to a blank ballot. The most preferable statement for liquid democracy in practice shall be determined by computational social choice theory, which is outside the scope of this paper.

5 Further Discussions

Statement policy. We initiate the study of statement voting and liquid democracy in this work. Our statement voting concept can be significantly extended to support much richer ballot statements, which opens a door for designing a new class of e-voting schemes. A natural question to ask is what type of statements are allowed. For correctness, the (deterministic) TallyProcess function should be a symmetric function in the sense that its output does not depend on the order of the ballots to be counted. Moreover, the voting statement has a maximum running time restriction to prevent DoS, and it should not depend on partial tally result. This is known as fairness. Namely, the statement execution cannot be conditional on the partial tally result at the moment when the ballot is counted. On the other hand, the statement can take input as external information oracles, such as News, Stock market, etc. When statement voting is integrated with a blockchain infrastructure, our scheme can be used to enable *offline voting* or *smart voting*. In particular, the voters may submit their statement ballot any time before the election on the blockchain; during the tally phase, the voter's ballots will be decrypted, and their statements will define their final votes based on the latest information provided by News oracles on the blockchain.

This line of research is far from being completed. We also remark that, voting policies can be heavily influenced by local legal and societal conditions. How to define “right” voting policy itself is a very interesting question. We believe our techniques here have the potential to help people to identify suitable voting policies which can further eliminate the barriers to democracy.

Trusted setup. Typically, trusted setup assumptions[3] are required for constructing UC-secure e-voting systems. Common Reference String (CRS) and Random Oracle (RO) are two popular choices in practice. If an e-voting system uses CRS, then we need to trust the party who generates the CRS, which, in our opinion, is a stronger assumption than believing no adversary can break a secure hash function, e.g., SHA3. Therefore, in this work, we realize our liquid democracy voting system in the RO model.

As a future direction, we will construct more solutions to liquid democracy. For example, an alternative approach is as follows: we first use multi-party computation (MPC) to generate a CRS; then we construct liquid democracy voting system by using the CRS. As argued above, we need to trust the parties who generate the CRS; here, at least one of the MPC players must be honest. This approach has previously been used for anonymous cryptocurrency; please see Ben-Sasson et al's recent effort [9]. We remark that, this approach might be problematic for cryptocurrency systems: typically a cryptocurrency system will last for many years and it is very difficult to ensure there is no attack on the CRS during this long time period. Interestingly, this limitation does not apply to liquid democracy voting systems. If there is an issue with the current CRS, we can use MPC to generate a new CRS.

[3] Most non-trivial functionalities (including the e-voting functionality) cannot be UC-realized in the plain model [18, 16, 20].

References

1 B. Adida. Helios: Web-based open-audit voting. In *USENIX Security*, pages 335–348, 2008.

2 D. Alger. Voting by proxy. *Public Choice*, 126(1):1–26, 2006.

3 J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Heidelberg, Aug. 2014.

4 J. Alwen, R. Ostrovsky, H.-S. Zhou, and V. Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 763–780. Springer, Heidelberg, Aug. 2015.

5 G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, Apr. 2012.

6 C. Baum, I. Damgård, and C. Orlandi. Publicly auditable secure multi-party computation. In M. Abdalla and R. D. Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 175–196. Springer, Heidelberg, Sept. 2014.

7 S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, Heidelberg, Apr. 2012.

8 E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

9 E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015.

10 D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, Heidelberg, Dec. 2012.

11 C. Blum and C. I. Zuber. Liquid democracy: Potentials, problems, and perspectives. *Journal of Political Philosophy*, 24(2):162–182, 2016.

12 Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Heidelberg, Aug. 2012.

13 Z. Brakerski and R. Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Heidelberg, Aug. 2016.

14 Z. Brakerski and V. Vaikuntanathan. Lattice-based FHE as secure as PKE. In M. Naor, editor, *ITCS 2014*, pages 1–12. ACM, Jan. 2014.

15 R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`.

16 R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

17 R. Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. `http://eprint.iacr.org/2003/239`.

18 R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, Aug. 2001.

19 R. Canetti, H. Krawczyk, and J. B. Nielsen. Relaxing chosen-ciphertext security. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 565–582. Springer, Heidelberg, Aug. 2003.

20 R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 68–86. Springer, Heidelberg, May 2003.

21 P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo. Beleniosrf: A non-interactive receipt-free electronic voting scheme. In *CCS '16*, pages 1614–1625, New York, NY, USA, 2016. ACM.

22 M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Malleable proof systems and applications. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300. Springer, Heidelberg, Apr. 2012.

23 D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora. Scantegrity: End-to-End Voter-Verifiable Optical- Scan Voting. *IEEE Security & Privacy Magazine*, 6(3):40–46, May 2008.

24 D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, Aug. 1993.

25 D. Chaum, P. Y. A. Ryan, and S. A. Schneider. A practical voter-verifiable election scheme. In S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, editors, *ESORICS 2005*, volume 3679 of *LNCS*, pages 118–139. Springer, Heidelberg, Sept. 2005.

26 I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, Sept. 2013.

27 I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, Aug. 2012.

28 B. Ford. Delegative democracy. 2002. `http://www.brynosaurus.com/deleg/deleg.pdf`.

29 C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, Aug. 2013.

30 J. Groth. Evaluating security of voting schemes in the universal composability framework. In M. Jakobsson, M. Yung, and J. Zhou, editors, *ACNS 04*, volume 3089 of *LNCS*, pages 46–60. Springer, Heidelberg, June 2004.

31 J. Groth. Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In M. Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 152–170. Springer, Heidelberg, Feb. 2004.

32 J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 253–280. Springer, Heidelberg, Apr. 2015.

33 S. Hardt and L. Lopes. Google votes: A liquid democracy experiment on a corporate social network. Technical Disclosure Commons, 2015. `http://www.tdcommons.org/dpubs_series/79`.

34 A. Kiayias, T. Zacharias, and B. Zhang. DEMOS-2: Scalable E2E verifiable elections without random oracles. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 352–363. ACM Press, Oct. 2015.

35 A. Kiayias, T. Zacharias, and B. Zhang. End-to-end verifiable elections in the standard model. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 468–498. Springer, Heidelberg, Apr. 2015.

36 A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.

37 A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan. Cloud-assisted multiparty computation from fully homomorphic encryption. Cryptology ePrint Archive, Report 2011/663, 2011. `http://eprint.iacr.org/2011/663`.

38 D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, Apr. 2012.

39 J. C. Miller. A program for direct and proxy voting in the legislative process. *Public Choice*, 7(1):107–113, 1969.

40 T. Moran and M. Naor. Receipt-free universally-verifiable voting with everlasting privacy. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 373–392. Springer, Heidelberg, Aug. 2006.

41 P. Mukherjee and D. Wichs. Two round multiparty computation via multi-key FHE. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

42 C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009.

43 C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 187–196. ACM Press, May 2008.

44 M. Prabhakaran and M. Rosulek. Rerandomizable RCCA encryption. In A. Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 517–534. Springer, Heidelberg, Aug. 2007.

45 O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

46 C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

47 D. Unruh and J. Müller-Quade. Universally composable incoercibility. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 411–428. Springer, Heidelberg, Aug. 2010.

A Preliminaries

A.1 The UC framework

Following Canetti's framework [16, 15], a protocol is represented as interactive Turing machines (ITMs), each of which represents the program to be run by a participant. Protocols that securely carry out a given task are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. The parties have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. Below we overview the model of protocol execution (called the *real-world model*), the ideal process, and the notion of protocol emulation.

The model for protocol execution. The model of computation consists of the parties running an instance of a protocol π, a network adversary $\mathcal{A}$ that controls the communication among the parties, and an environment $\mathcal{Z}$ that controls the inputs to the parties and sees their outputs. The execution consists of a sequence of *activations,* where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some other ITM) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete, the participant whose tape was written on is activated next.

Let $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z, r)$ denote the output of the environment $\mathcal{Z}$ when interacting with parties running protocol π on security parameter λ, input z and random input $r = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, r_2, ...$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$; $r_{\mathcal{A}}$ for $\mathcal{A}$, r_i for party P_i). Let $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the random variable describing $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, r)$ when r is uniformly chosen. Let $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$.

Ideal functionalities and ideal protocols. Security of protocols is defined via comparing the protocol execution to an *ideal protocol* for carrying out the task at hand. A key ingredient in the ideal protocol is the *ideal functionality* that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM (representing a "trusted party") that interacts with the parties and the adversary. More specifically, in the ideal protocol for functionality $\mathcal{F}$ all parties simply hand their inputs to an ITM instance running $\mathcal{F}$.

Securely realizing an ideal functionality. We say that a protocol π *emulates* protocol ϕ if for any network adversary $\mathcal{A}$ there exists an adversary (also known as simulator) $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with A and parties running π, or it is interacting with $\mathcal{S}$ and parties running ϕ. This means that, from the point of view of the environment, running protocol π is "just as good" as interacting with ϕ. We say that π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol for $\mathcal{F}$. More precise definitions follow. A distribution ensemble is called *binary* if it consists of distributions over $\{0, 1\}$.

▶ **Definition 2.** Let π and ϕ be protocols, and $\mathcal{F}$ be an ideal functionality. We say that π UC-emulates ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have $\mathsf{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. We say that π UC-realizes $\mathcal{F}$ if π UC-emulates the ideal protocol for functionality $\mathcal{F}$.

Hybrid protocols. Hybrid protocols are protocols where, in addition to communicating as usual as in the standard model of execution, the parties also have access to (multiple copies of) an ideal functionality. Hybrid protocols represent protocols that use idealizations of underlying primitives, or alternatively make *trust assumptions* on the underlying network. They are also instrumental in stating the universal composition theorem. Specifically, in an $\mathcal{F}$-hybrid protocol (i.e., in a hybrid protocol with access to an ideal functionality $\mathcal{F}$), the parties may give inputs to and receive outputs from an unbounded number of copies of $\mathcal{F}$. The definition of a protocol securely realizing an ideal functionality is extended to hybrid protocols in the natural way.

A.2 Ideal functionalities

A.2.1 Bulletin board functionality

The public bulletin board (BB) is modeled as a global functionality $\bar{\mathcal{G}}_{\mathrm{BB}}$. Formal description can be found in Fig. 7. The functionality is parameterized with a predicate $\mathsf{Validate}$ that ensures all the newly posted messages are consistent with the existing BB content w.r.t. $\mathsf{Validate}$. Any party can use $(\textsc{submit}, \mathsf{sid}, \mathsf{msg})$ and $(\textsc{read}, \mathsf{sid})$ to write/read the BB. We remark that our $\bar{\mathcal{G}}_{\mathrm{BB}}$ can be much simplified version of the global public ledger functionality $\bar{\mathcal{G}}_{\textsc{ledger}}$ recently defined by Kiayias et al [36].

Functionality $\bar{\mathcal{G}}_{\mathrm{BB}}$

The shared functionality $\bar{\mathcal{G}}_{\mathrm{BB}}$ is globally available to all the parties and the adversary $\mathcal{S}$. It is parameterized with a predicate $\mathsf{Validate}$, and variable state. Initially, $\mathsf{state} := \varepsilon$.

- Upon receiving $(\textsc{submit}, \mathsf{sid}, \mathsf{msg})$ from a party P or the adversary $\mathcal{S}$, if $\mathsf{Validate}(\mathsf{state}, \mathsf{msg}) = 1$, then set $\mathsf{state} := \mathsf{state}||\mathsf{msg}$.
- Upon receiving $(\textsc{read}, \mathsf{sid})$ from a party P or the adversary $\mathcal{S}$, return $(\textsc{read}, \mathsf{sid}, \mathsf{state})$ to the requestor.

Figure 7 The public bulletin board functionality.

A.2.2 Certificate functionality

We present the multi-session version of certificate functionality following the modeling of [17]. The multi-session certificate functionality $\widehat{\mathcal{F}}_{\textsc{Cert}}$ can provide direct binding between a signature for a message and the identity of the corresponding signer. This corresponds to providing signatures accompanied by "certificates" that bind the verification process to the signers' identities. For completeness, we recap $\widehat{\mathcal{F}}_{\textsc{Cert}}$ in Fig. 8.

Functionality $\widehat{\mathcal{F}}_{\textsc{Cert}}$

The functionality $\widehat{\mathcal{F}}_{\textsc{Cert}}$ interacts with a set of signers $\{\mathsf{S}_1, \dots, \mathsf{S}_k\}$, and a set of verifiers $\{\mathsf{R}_1, \dots, \mathsf{R}_n\}$, and the adversary $\mathcal{S}$.

- Upon receiving $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, m)$ from a signer $P \in \{\mathsf{S}_1, \dots, \mathsf{S}_k\}$, verify that $\mathsf{ssid} = (P, \mathsf{ssid}')$ for some ssid'. If not, ignore the request. Otherwise, send $(\textsc{SignNotify}, \mathsf{sid}, \mathsf{ssid}, m)$ to the adversary $\mathcal{S}$. Upon receiving $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, m, \sigma)$ from $\mathcal{S}$, verify that no entry $(\mathsf{ssid}, m, \sigma, 0)$ is recorded. If it is, then return $(\textsc{Error})$ to P and halt. Else, return $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, m, \sigma)$ to P, and record the entry $(\mathsf{ssid}, m, \sigma, 1)$.
- Upon receiving $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, m, \sigma)$ from any party $P \in \{\mathsf{R}_1, \dots, \mathsf{R}_n\}$, send $(\textsc{VerifyNotify}, \mathsf{sid}, \mathsf{ssid}, m)$ to the adversary $\mathcal{S}$. Upon receiving $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, m, b^*)$ from $\mathcal{S}$, do:
  - If $(\mathsf{ssid}, m, \sigma, 1)$ is recorded then set $b := 1$.
  - Else, if the signer of subsession ssid is not corrupted, and no entry $(\mathsf{ssid}, m, \cdot, 1)$ is recorded, then set $b = 0$ and record the entry $(\mathsf{ssid}, m, \sigma, 0)$.
  - Else, if there is an entry $(\mathsf{ssid}, m, \sigma, b')$ recorded, then set $b := b'$.
  - Else, set $b := b^*$, and record the entry $(\mathsf{ssid}, m, \sigma, b^*)$.

  Output $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, m, b)$ to P.

Figure 8 The multi-session functionality for certificate.

A.2.3 Non-interactive zero-knowledge proofs/arguments

Here we briefly introduce non-interactive zero-knowledge (NIZK) schemes in the Random Oracle (RO) model. Let $\mathcal{R}$ be an efficiently computable binary relation. For pairs $(x, w) \in \mathcal{R}$ we call x the statement and w the witness. Let $\mathcal{L}_{\mathcal{R}}$ be the language consisting of statements in $\mathcal{R}$, i.e. $\mathcal{L}_{\mathcal{R}} = \{x | \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$. An NIZK scheme includes following algorithms: a PPT algorithm Prov that takes as input $(x, w) \in \mathcal{R}$ and outputs a proof π; a polynomial time algorithm Verify takes as input (x, π) and outputs 1 if the proof is valid and 0 otherwise.

▶ **Definition 3** (NIZK Proof of Membership in the RO Model). $\mathsf{NIZK}^{\mathrm{RO}}_{\mathcal{R}}.\{\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext}\}$ is an NIZK Proof of Membership scheme for the relation $\mathcal{R}$ if the following properties hold:

- *Completeness:* For any $(x, w) \in \mathcal{R}$,

$$\Pr\left[\ \zeta \leftarrow \{0,1\}^\lambda; \pi \leftarrow \mathsf{Prov}^{\mathrm{RO}}(x, w; \zeta) : \mathsf{Verify}^{\mathrm{RO}}(x, \pi) = 0\ \right] \leq \mathsf{negl}(\lambda).$$

- *Zero-knowledge:* If for any PPT distinguisher $\mathcal{A}$ we have

$$\left|\ \Pr[\mathcal{A}^{\mathrm{RO},\mathcal{O}_1}(1^\lambda) = 1] - \Pr[\mathcal{A}^{\mathrm{RO},\mathcal{O}_2}(1^\lambda) = 1]\ \right| \leq \mathsf{negl}(\lambda).$$

  The oracles are defined as follows: $\mathcal{O}_1$ on query $(x, w) \in \mathcal{R}$ returns π, where $(\pi, aux) \leftarrow \mathsf{Sim}^{\mathrm{RO}}(x)$; $\mathcal{O}_2$ on query $(x, w) \in \mathcal{R}$ returns π, where $\pi \leftarrow \mathsf{Prov}^{\mathrm{RO}}(x, w; \zeta)$ and $\zeta \leftarrow \{0,1\}^\lambda$.
- *Soundness:* For all PPT adversary $\mathcal{A}$,

$$\Pr\left[\ (x, \pi) \leftarrow \mathcal{A}^{\mathrm{RO}}(1^\lambda) : x \notin \mathcal{L}_R \wedge \mathsf{Verify}^{\mathrm{RO}}(x, \pi) = 1\ \right] \leq \mathsf{negl}(\lambda).$$

▶ **Definition 4** (NIZK Proof of Knowledge in the RO Model)**.** $\mathsf{NIZK}_{\mathcal{R}}^{\mathrm{RO}}.\{\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext}\}$ is an NIZK Proof of Knowledge scheme for the relation $\mathcal{R}$ if the completeness, zero-knowledge, and extraction properties hold, where the extraction is defined as follows.

- *Extractability:* For all PPT adversary $\mathcal{A}$,

$$\Pr\left[\ (x, \pi) \leftarrow \mathcal{A}^{\mathrm{RO}}(1^\lambda); w \leftarrow \mathsf{Ext}^{\mathrm{RO}}(x, \pi) : (x, w) \in \mathcal{R} \text{ if } \mathsf{Verify}^{\mathrm{RO}}(x, \pi) = 1\ \right] \geq 1 - \mathsf{negl}(\lambda).$$

We need non-interactive zero-knowledge proofs/arguments of knowledge and non-interactive zero-knowledge proofs/arguments of membership. For simplicity, we will drop RO from the superscript if the context is clear.

We use $\mathsf{NIZK}_{\mathcal{R}_i}.\mathsf{Verify}$ and $\mathsf{NIZK}_{\mathcal{R}_i}.\mathsf{Sim}$) to denote the corresponding verification algorithm and simulator, respectively.

B Homomorphic Encryption based construction

In this section, we present a (key-homomorphic) threshold fully homomorphic encryption (FHE) based scheme. It is organised as follows. In Appendix B.1, we provide the syntax and security definition of a key-homomorphic threshold FHE. In Appendix B.4, we provide an instantiation of the TFHE via the GSW scheme (Cf. Appendix B.6).

A key-homomorphic public key encryption scheme allows the user to deterministically combine several public keys $\mathtt{pk}_1, \ldots, \mathtt{pk}_n$ into a combined public key $\mathtt{pk}$; meanwhile, the corresponding secret keys $\mathtt{sk}_1, \ldots, \mathtt{sk}_n$ can be combined into the secret key $\mathtt{sk}$ for $\mathtt{pk}$. This property can enable efficient distributed key generation. In a nutshell, the scheme works as follows. During the preparation phase, each trustee $\mathsf{T}_j \in \mathbb{T}$ generates a public key $\mathtt{pk}_j$ and posts it on the $\bar{\mathcal{G}}_{\mathrm{BB}}$. The voters $\mathsf{V}_i \in \mathbb{V}$ can then combine the posted $\mathtt{pk}_j$'s to the election public key $\mathtt{pk}$. During the ballot casting phase, the voters $\mathsf{V}_i \in \mathbb{V}$ submit their encrypted statement to the $\bar{\mathcal{G}}_{\mathrm{BB}}$. After that, all the parties can evaluate the (public deterministic) $\mathsf{TallyProcess}$ circuit over the encrypted data. During the tally phase, the trustees $\mathsf{T}_j \in \mathbb{T}$ then jointly decrypt the final tally ciphertext(s) to reveal the election outcome. We will now introduce the main primitive, publicly-evaluable key-homomorphic *threshold fully homomorphic encryption* (TFHE).

B.1 Key-homomorphic threshold fully homomorphic encryption

A publicly evaluable key-homomorphic threshold fully homomorphic encryption scheme TFHE consists of a tuple of algorithms: $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{CombinePK}, \mathsf{CombineSK}, \mathsf{ShareDec}, \mathsf{ShareCombine})$ as follows.

- $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$. The algorithm Setup takes input as the security parameter λ, and outputs public parameters param. All the other algorithms implicitly take param as input.
- $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{Keygen}(\mathsf{param})$. The algorithm Keygen takes input as the public parameter param, and outputs a public key $\mathtt{pk}$, a secret key $\mathtt{sk}$.
- $c \leftarrow \mathsf{Enc}(\mathtt{pk}, m)$. The algorithm Enc takes input as the public key $\mathtt{pk}$ and the message m, and outputs the ciphertext c.
- $c' := \mathsf{Eval}(\mathtt{pk}, \mathcal{F}, c_1, \ldots, c_n)$. The algorithm Eval takes input as the public/evaluation key $\mathtt{pk}$, the description of the evaluation function (circuit) $\mathcal{F}$, and a set of ciphertexts $c_1, \ldots, c_n$, and outputs the result ciphertext c'.
- $m \leftarrow \mathsf{Dec}(\mathtt{sk}, c)$. The algorithm Dec takes input as the secret key $\mathtt{sk}$ and a ciphertext c, and outputs the decrypted plaintext m.
- $\mathtt{pk} := \mathsf{CombinePK}(\mathtt{pk}_1, \ldots, \mathtt{pk}_k)$. The algorithm CombinePK takes input as a set of public keys $(\mathtt{pk}_1, \ldots, \mathtt{pk}_k)$, and outputs a combined public key $\mathtt{pk}$.
- $\mathtt{sk} \leftarrow \mathsf{CombineSK}(\mathtt{sk}_1, \ldots, \mathtt{sk}_k)$. The algorithm CombineSK takes input as a set of secret key $(\mathtt{sk}_1, \ldots, \mathtt{sk}_k)$, and outputs combined secret key $\mathtt{sk}$.
- $\mu_i \leftarrow \mathsf{ShareDec}(\mathtt{sk}_i, c)$. The algorithm ShareDec takes input as the secret key $\mathtt{sk}_i$ and a ciphertext c, and outputs a decryption share μ_i.
- $m \leftarrow \mathsf{ShareCombine}(c, \mu_1, \ldots, \mu_k)$. The algorithm ShareCombine takes input as a ciphertext c and k decryption shares $(\mu_1, \ldots, \mu_k)$, and outputs a plaintext m.
- $c' \leftarrow \mathsf{Trans}(c, \{\mathtt{sk}_i\}_{i \in [k] \setminus \{j\}})$. The algorithm Trans takes input as a ciphertext $c \leftarrow \mathsf{TFHE.Enc}(\mathtt{pk}_j, m)$ and a set of secret keys $\{\mathtt{sk}_i\}_{i \in [k] \setminus \{j\}}$, and outputs a ciphertext c'.
- $\{\mu_j\}_{j \in [k] \setminus \mathcal{I}} \leftarrow \mathsf{SimShareDec}(c, m, \{\mu_i\}_{i \in \mathcal{I}})$. The algorithm SimShareDec takes as input a ciphertext c, a plaintext m, and a set of decryption shares $\{\mu_i\}_{i \in \mathcal{I}}$ and outputs a set of decryption shares $\{\mu_j\}_{j \in [k] \setminus \mathcal{I}}$. Here $\mathcal{I} \subsetneq [k]$.

▶ **Definition 5.** We say $\mathsf{TFHE} = \{\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{CombinePK}, \mathsf{CombineSK}, \mathsf{ShareDec}, \mathsf{ShareCombine}\}$ is a *secure* key-homomorphic threshold fully homomorphic encryption if the following properties hold:

- Key combination correctness:
  If $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$ are all valid key pairs, $\mathtt{pk} := \mathsf{TFHE.CombinePK}(\{\mathtt{pk}_i\}_{i\in[k]})$ and $\mathtt{sk} := \mathsf{TFHE.CombineSK}(\{\mathtt{sk}_i\}_{i\in[k]})$, then $(\mathtt{pk}, \mathtt{sk})$ is also a valid key pair.
  For all ciphertext $c \in \mathcal{C}_{\mathtt{pk}}$, where $\mathcal{C}_{\mathtt{pk}}$ is the ciphertext-space defined by $\mathtt{pk}$, we have
  $$\mathsf{TFHE.Dec}(\mathtt{sk}, c) = \mathsf{TFHE.ShareCombine}(c, \mathsf{TFHE.ShareDec}(\mathtt{sk}_1, c), \ldots, \mathsf{TFHE.ShareDec}(\mathtt{sk}_k, c)) \ .$$
- Ciphertext transformative indistinguishability: We say that a TFHE scheme achieves *ciphertext transformative indistinguishability*, if for all message m, for any $j \in [k]$, there exists a PPT algorithm Trans such that
  $$\big(\mathsf{param}, \mathsf{TFHE.Trans}(c, \{\mathtt{sk}_i\}_{i\in[k]\setminus\{j\}})\big) \approx \big(\mathsf{param}, \mathsf{TFHE.Enc}(\mathtt{pk}, m)\big)$$
  where $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$ are all valid key pairs, $\mathtt{pk} := \mathsf{TFHE.CombinePK}(\{\mathtt{pk}_i\}_{i\in[k]})$ and $\mathtt{sk} := \mathsf{TFHE.CombineSK}(\{\mathtt{sk}_i\}_{i\in[k]})$.
- Share-simulation indistinguishability: We say TFHE scheme achieves *share-simulation indistinguishability* if there exists a PPT simulator $\mathsf{SimShareDec}$ such that for all valid key pairs $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$, all subsets $\mathcal{I} \subsetneq [k]$, all message m, the following two distributions are computationally indistinguishable:
  $$\big(\mathsf{param}, c, \mathsf{SimShareDec}(c, m, \{\mu_i\}_{i\in\mathcal{I}})\big) \approx \big(\mathsf{param}, c, \{\mu_j\}_{j\in[k]\setminus\mathcal{I}}\big)$$
  where $\mathsf{param} \leftarrow \mathsf{TFHE.Setup}(1^\lambda)$, $c \leftarrow \mathsf{TFHE.Enc}(\mathtt{pk}, m)$ and $\mu_j \leftarrow \mathsf{TFHE.ShareDec}(\mathtt{sk}_j, c)$ for $j \in [k] \setminus \mathcal{I}$.

The above TFHE syntax is adopted from Lopez-Alt *et al.* [37, 5], with the following modification: we use $\mathtt{pk}$ as the evaluation key. This minor change will allow us to evulate the ciphertexts publicly. In [37, 5], only distinguished players (i.e., the ones with secret keys) after a joint protocol, can obtain the evaluation key. Note that, the TFHE scheme can be instantiated via [29].

B.2 Protocol description

In this section, we formally describe our TFHE-based construction for statement voting. The protocol is designed in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\mathrm{CERT}}\}$-hybrid world and it consists of three phases: preparation, ballot casting, and tally. For the sake of notation simplicity, we omit the processes of filtering invalid messages on $\bar{\mathcal{G}}_{\mathrm{BB}}$. In practice, $\bar{\mathcal{G}}_{\mathrm{BB}}$ contains many messages with invalid signatures, and all those messages should be ignored.

B.2.1 Preparation phase

As depicted in Figure 9, in the preparation phase, each trustee T_j, first picks a randomness generates α_j and generates a partial public key using $(\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) \leftarrow \mathsf{TFHE.Keygen}(\mathsf{param}; \alpha_j)$. It then generates an NIZK proof
$$\pi_j^{(1)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_1}\left\{\ (\overline{\mathtt{pk}}_j), (\alpha_j, \overline{\mathtt{sk}}_j) : (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{TFHE.Keygen}(\mathsf{param}; \alpha_j)\ \right\}$$
to show that this process is executed correctly; namely, it shows knowledge of $(\alpha_j, \overline{\mathtt{sk}}_j)$ w.r.t. to the generated partial public key $\overline{\mathtt{pk}}_j$. It then signs and posts $(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Preparation

Upon receiving ($\textsc{InitialTrustee}, \mathsf{sid}$) from the environment $\mathcal{Z}$, the trustee T_j, $j \in [k]$, operates as the follows:

- Generate $(\overline{\mathsf{pk}}_j, \overline{\mathsf{sk}}_j) \leftarrow \mathsf{TFHE.Keygen}(\mathsf{param}; \alpha_j)$ where α_j is the fresh randomness, and then compute

$$\pi_j^{(1)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_1}\left\{ \ (\overline{\mathsf{pk}}_j), (\alpha_j, \overline{\mathsf{sk}}_j) : (\overline{\mathsf{pk}}_j, \overline{\mathsf{sk}}_j) = \mathsf{TFHE.Keygen}(\mathsf{param}; \alpha_j) \ \right\}$$

- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathsf{pk}}_j, \pi_j^{(1)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and receives $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathsf{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (\overline{\mathsf{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)} \rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Figure 9 TFHE based statement voting scheme $\Pi_{\text{FHE-SV}}$ in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part I).

B.2.2 Ballot casting phase

As depicted in Figure 10, in the ballot casting phase, each voter V_i, $i \in [n]$ first fetches the partial public keys $\{\overline{\mathsf{pk}}_j\}_{j\in[k]}$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. After checking their corresponding NIZK proofs, the voter V_i combines them to the election public key $\mathsf{pk} := \mathsf{TFHE.CombinePK}(\{\overline{\mathsf{pk}}_j\}_{j=1}^k)$. V_i then encrypts his ballot (V_i, s_i) as $c_i = \mathsf{TFHE.Enc}(\mathsf{pk}, (\mathsf{V}_i, s_i))$. Here V_i is abused as the voter's PID and s_i is her statement. The voter then posts the ciphertext c_i on the $\bar{\mathcal{G}}_{\mathrm{BB}}$ together with the corresponding NIZK proof showing that c_i is indeed generated by the voter V_i.

Ballot Casting

Upon receiving $(\textsc{Cast}, \mathsf{sid}, s_i)$ from the environment $\mathcal{Z}$, the voter V_i, $i \in [n]$ operates as the follows:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. If $\left\{ \langle \mathsf{ssid}, (\overline{\mathsf{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)} \rangle \right\}_{j\in[k]}$ is contained in state, then for $j \in [k]$, send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathsf{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathsf{pk}}_j, \pi_j^{(1)}), b_j^{(1)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; If $\prod_{j=1}^k b_j^{(1)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Verify}(\overline{\mathsf{pk}}_j, \pi_j^{(1)}) = 1$ for $j \in [k]$. If any of the checks is invalid, halt.
- Compute and store $\mathsf{pk} := \mathsf{TFHE.CombinePK}(\{\overline{\mathsf{pk}}_j\}_{j=1}^k)$.
- Encrypt $c_i \leftarrow \mathsf{TFHE.Enc}(\mathsf{pk}, (\mathsf{V}_i, s_i); \beta_i)$ where β_i is the fresh randomness, and then compute

$$\pi_i^{(2)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_2}\left\{ \ (\mathsf{pk}, c_i), (\mathsf{V}_i, s_i, \beta_i) : c_i = \mathsf{TFHE.Enc}(\mathsf{pk}, (\mathsf{V}_i, s_i); \beta_i) \ \right\}$$

- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (c_i, \pi_i^{(2)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{V}_i, \mathsf{ssid}')$ for some ssid', and receive $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (c_i, \pi_i^{(2)}), \sigma_i^{(2)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (c_i, \pi_i^{(2)}), \sigma_i^{(2)} \rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Figure 10 TFHE based statement voting scheme $\Pi_{\text{FHE-SV}}$ in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part II).

B.2.3 Tally phase

The tally phase is depicted in Figure 11. The trustee $\mathsf{T}_j \in \mathbb{T}$, $j \in [k]$ fetches the posted encrypted ballots $\{c_i\}_{i\in[n]}$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. It checks the corresponding NIZK proofs and removes the invalid ones. Each of the trustees $\mathsf{T}_j \in \mathbb{T}$ then evaluates the $\mathsf{TallyProcess}$ circuit as $c := \mathsf{TFHE.Eval}(\mathtt{pk}, \mathsf{TallyProcess}, c_1, \ldots, c_n)$. After that, all the trustees jointly decrypt c to the final tally τ, attached with necessary NIZK proofs. Finally, all the voters $\mathsf{V}_i \in \mathbb{V}$ can read the tally result τ from $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Tally

Upon receiving $(\textsc{Tally}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the trustee T_j, where $j \in [k]$, operates as the follows:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. If $\left\{\langle \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)}\rangle\right\}_{j\in[k]}$ is contained in state, then for $j \in [k]$, send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), b_j^{(1)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; If $\prod_{j=1}^{k} b_j^{(1)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Verify}(\overline{\mathtt{pk}}_j, \pi_j^{(1)}) = 1$ for $j \in [k]$. If any of the checks is invalid, halt.
- Compute $\mathtt{pk} \leftarrow \mathsf{TFHE.CombinePK}(\{\overline{\mathtt{pk}}_j\}_{j=1}^{k})$.
- For $i \in [n]$, if $\langle \mathsf{ssid}, (c_i, \pi_i^{(2)}), \sigma_i^{(2)}\rangle$ is contained in state, then send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (c_i, \pi_i^{(2)}), \sigma_i^{(2)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (c_i, \pi_i^{(2)}), b_i^{(2)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; if $b_i^{(2)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\overline{\mathtt{pk}}, c_i), \pi_i^{(2)}) = 1$. If any of the above checks is invalid, reset $c_i := \perp$.
- Compute $c := \mathsf{TFHE.Eval}(\mathtt{pk}, \mathsf{TallyProcess}, c_1, \ldots, c_n)$.
- Compute $\overline{\tau}_j \leftarrow \mathsf{TFHE.ShareDec}(\overline{\mathtt{sk}}_j, c)$ together with

$$\pi_j^{(3)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_3}\left\{\begin{array}{l} (c, \overline{\tau}_j, \overline{\mathtt{pk}}_j), (\overline{\mathtt{sk}}_j, \alpha_j) : \\ \ (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{TFHE.Keygen}(\mathsf{param}; \alpha_j) \\ \ \wedge\ \overline{\tau}_j = \mathsf{TFHE.ShareDec}(\overline{\mathtt{sk}}_j, c) \end{array}\right\}$$

- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (\overline{\tau}_j, \pi_j^{(3)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and receives $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (\overline{\tau}_j, \pi_j^{(3)}), \sigma_j^{(3)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (\overline{\tau}_j, \pi_j^{(3)}), \sigma_j^{(3)}\rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Upon receiving $(\textsc{ReadResult}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the voter V_i, $i \in [n]$ operates as the follows:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. If $\left\{\langle \mathsf{ssid}, (\overline{\tau}_j, \pi_j^{(3)}), \sigma_j^{(3)}\rangle\right\}_{j\in[k]}$ is contained in state, then for $j \in [k]$, send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (\overline{\tau}_j, \pi_j^{(3)}), \sigma_j^{(3)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (\overline{\tau}_j, \pi_j^{(3)}), b_j^{(3)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; If $\prod_{j=1}^{k} b_j^{(3)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Verify}((c, \overline{\tau}_j, \overline{\mathtt{pk}}_j), \pi_j^{(3)}) = 1$ for $j \in [k]$. If any of the checks is invalid, halt.
- Compute $\tau \leftarrow \mathsf{TFHE.ShareCombine}(\{\overline{\tau}_j\}_{j=1}^{k})$.
- Return $(\textsc{ReadResultReturn}, \mathsf{sid}, \tau)$ to the environment $\mathcal{Z}$.

Figure 11 TFHE based statement voting scheme $\Pi_{\text{FHE-SV}}$ in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part III).

B.3 Security

We recap a stronger statement voting functionality $\mathcal{F}_{\mathrm{SV}}$ in Fig. 12 and prove the following the theorem.

▶ **Theorem 6.** *Protocol* $\Pi_{\text{FHE-SV}}$ *described in Figure 9, Figure 10 and Figure 11 UC-realizes* $\mathcal{F}_{\text{SV}}$ *in the* $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}\}$*-hybrid world against static corruption.*

Functionality $\mathcal{F}_{\text{SV}}$

The functionality $\mathcal{F}_{\text{SV}}$ interacts with a set of voters $\mathbb{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, a set of trustees $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$, and the adversary $\mathcal{S}$. Let $\mathbb{V}_{\mathsf{honest}}$, $\mathbb{V}_{\mathsf{corrupt}}$ and $\mathbb{T}_{\mathsf{honest}}$, $\mathbb{T}_{\mathsf{corrupt}}$ denote the set of honest/corrupt voters and trustees, respectively.
Functionality $\mathcal{F}_{\text{SV}}$ is parameterized by an algorithm $\mathsf{TallyProcess}$, and variables *result*, T_1, T_2, and B_i for all $i \in [n]$.
Initially, set $\mathit{result} := \emptyset$, $T_1 := \emptyset$, $T_2 := \emptyset$; for $i \in [n]$, set $B_i := \emptyset$.

Preparation:

1. Upon receiving input $(\textsc{InitialTrustee}, \mathsf{sid})$ from the trustee $\mathsf{T}_j \in \mathbb{T}$, set $T_1 := T_1 \cup \{\mathsf{T}_j\}$, and send a notification message $(\textsc{InitialTrusteeNotify}, \mathsf{sid}, \mathsf{T}_j)$ to the adversary $\mathcal{S}$.

Ballot Casting:

1. Upon receiving input $(\textsc{Cast}, \mathsf{sid}, s_i)$ from the voter $\mathsf{V}_i \in \mathbb{V}$, if $|T_1| < k$, ignore the input. Otherwise, record $B_i := (\mathsf{V}_i, s_i)$; send a message $(\textsc{CastNotify}, \mathsf{sid}, \mathsf{V}_i)$ to the adversary $\mathcal{S}$. If $|\mathbb{T}_{\mathsf{corrupt}}| = k$, then additionally send a message $(\textsc{Leak}, \mathsf{sid}, \mathsf{V}_i, B_i)$ to the adversary $\mathcal{S}$.

Tally:

1. Upon receiving input $(\textsc{Tally}, \mathsf{sid})$ from the trustee $\mathsf{T}_j \in \mathbb{T}$, if $\mathsf{T}_j \notin T_2$ then set $T_2 := T_2 \cup \{\mathsf{T}_j\}$.
   Send a notification message $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_j)$ to $\mathcal{S}$.
   If $|T_2 \cap \mathbb{T}_{\mathsf{honest}}| + |\mathbb{T}_{\mathsf{corrupt}}| = k$, send $(\textsc{Leak}, \mathsf{sid}, \mathsf{TallyProcess}(B_1, \ldots, B_n))$ to $\mathcal{S}$.
   If $|T_2| = k$, compute $\mathit{result} \leftarrow \mathsf{TallyProcess}(B_1, \ldots, B_n)$.
2. Upon receiving input $(\textsc{ReadResult}, \mathsf{sid})$ from a voter $\mathsf{V}_i \in \mathbb{V}$, if $\mathit{result} = \emptyset$, ignore the input. Else, return $(\textsc{ResultReturn}, \mathsf{sid}, \mathit{result})$ to V_i.

Figure 12 The statement voting functionality $\mathcal{F}_{\text{SV}}$.

Proof. To prove the theorem, we construct a simulator $\mathcal{S}$ such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}}_{\Pi_{\text{FHE-SV}}, \mathcal{A}, \mathcal{Z}}$ where the parties $\mathbb{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ and $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$ run protocol $\Pi_{\text{FHE-SV}}$ in the $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}\}$-hybrid world and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\text{BB}}}_{\mathcal{F}_{\text{SV}}, \mathcal{S}, \mathcal{Z}}$ where the parties interact with functionality $\mathcal{F}_{\text{SV}}$ in the $\bar{\mathcal{G}}_{\text{BB}}$-hybrid model and corrupted parties are controlled by the simulator $\mathcal{S}$. Let $\mathbb{V}_{\mathsf{corrupt}} \subseteq \mathbb{V}$ and $\mathbb{T}_{\mathsf{corrupt}} \subseteq \mathbb{T}$ be the set of corrupted voters and trustees, respectively. We consider following cases.

Case 1: $0 \leq |\mathbb{V}_{\mathsf{corrupt}}| < n \ \wedge \ 0 \leq |\mathbb{T}_{\mathsf{corrupt}}| < k$.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates honest voters $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, honest trustees $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$ and functionalities $\widehat{\mathcal{F}}_{\text{CERT}}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- In the preparation phase:
  - Upon receiving $(\textsc{InitialTrusteeNotify}, \mathsf{sid}, \mathsf{T}_j)$ from the external $\mathcal{F}_{\text{SV}}$ for an honest trustee $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j, following the protocol $\Pi_{\text{FHE-SV}}$ as if T_j receives $(\textsc{InitialTrustee}, \mathsf{sid})$ from $\mathcal{Z}$.

  - Monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$, when a valid $(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, use $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Ext}^{\mathrm{RO}}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corresponding secret key $\overline{\mathtt{sk}}_j$.
- In the ballot casting phase:
  - Upon receiving $(\textsc{CastNotify}, \mathsf{sid}, \mathsf{V}_i)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest voter $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ creates $c_i \leftarrow \mathsf{TFHE.Enc}(\mathtt{pk}, 0)$. It then uses $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Sim}$ to simulate the corresponding proofs $\pi_i^{(2)}$. The simulator $\mathcal{S}$ then follows the protocol to post $(c_i, \pi_i^{(2)})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.
  - Monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$, when a valid $(c_i, \pi_i^{(2)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted voter $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, uses the extracted $\left\{\overline{\mathtt{sk}}_j\right\}_{j\in[k]}$ to decrypt c_i to (V_i, s_i). The simulator $\mathcal{S}$ then acts as V_i to send $(\textsc{Cast}, \mathsf{sid}, s_i)$ to $\mathcal{F}_{\mathrm{SV}}$.
- In the tally phase:
  - Upon receiving $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_j)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest trustee $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$, if $\overline{\tau}_j$ are not defined yet, the $\mathcal{S}$ acts as T_j, following the protocol $\Pi_{\text{FHE-SV}}$ as if T_j receives $(\textsc{Tally}, \mathsf{sid})$ from $\mathcal{Z}$. $\mathcal{S}$ then adds j to $\mathcal{J}$, where $\mathcal{J}$ is initially empty. If $\overline{\tau}_j$ is defined, $\mathcal{S}$ uses $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Sim}$ to simulate the corresponding proof $\pi_j^{(3)}$. It then follows the protocol to post $(\overline{\tau}_j, \pi_j^{(3)})$ on the $\bar{\mathcal{G}}_{\mathrm{BB}}$.
  - Upon receiving $(\textsc{Leak}, \mathsf{sid}, \tau)$ from the external $\mathcal{F}_{\mathrm{SV}}$, the simulator $\mathcal{S}$ uses the extracted secret key $\overline{\mathtt{sk}}_j$ to compute $\overline{\tau}_j \leftarrow \mathsf{TFHE.ShareDec}(\overline{\mathtt{sk}}_j, c)$ for all the corrupted trustees $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$. It then adds all the indices of the corrupted trustees to $\mathcal{J}$. The simulator $\mathcal{S}$ computes $\{\overline{\tau}_j\}_{j\in[k]\setminus\mathcal{J}} \leftarrow \mathsf{SimShareDec}(c, \tau, \{\overline{\tau}_i\}_{i\in\mathcal{J}})$.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_4$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\mathrm{CERT}}}_{\Pi_{\text{FHE-SV}}, \mathcal{A}, \mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that $\mathcal{H}_1$ runs $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Ext}^{\mathrm{RO}}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corrupted trustee's secret key $\overline{\mathtt{sk}}_j$. $\mathcal{H}_1$ halt if the extraction fails.

▶ Claim 7. $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable.

Proof. According to Def. 4, the probability $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Ext}^{\mathrm{RO}}$ extraction fails (a.k.a. knowledge error) is negligible, so the probability that any adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$ is $\mathsf{negl}(\lambda)$. ◀

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except the following: During the tally phase, uses the extracted $\mathtt{sk}_j$ from Hybrid $\mathcal{H}_1$ to decrypt each ciphertext, and the last honest trustee's message shares of each ciphertext are calculated by TFHE.SimShareDec instead of using TFHE.ShareDec.

▶ Claim 8. $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable.

Proof. By the share-simulation indistinguishability of the underlying TFHE scheme, the distribution of the simulated decryption share(s) are computationally indistinguishable to the real ones. Moreover, by soundness of

$$\mathsf{NIZK}_{\mathcal{R}_3}\left\{\begin{array}{l}(c, \overline{\tau}_j, \overline{\mathtt{pk}}_j), (\overline{\mathtt{sk}}_j, \alpha_j) : \\ (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{TFHE.Keygen}(\mathsf{param}; \alpha_j) \ \wedge \ \overline{\tau}_j = \mathsf{TFHE.ShareDec}(\overline{\mathtt{sk}}_j, c)\end{array}\right\}$$

the corrupted trustees have negligible probability to post an invalid decryption share that is different from $\overline{\tau}_j \leftarrow \mathsf{TFHE.ShareDec}(\overline{\mathtt{sk}}_j, c)$. Therefore, the adversary's advantage of distinguishing $\mathcal{H}_2$ from $\mathcal{H}_1$ is $\mathsf{negl}(\lambda)$. ◀

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except the followings. During the vote phase, $\mathcal{H}_3$ uses $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Sim}$ to simulate $\pi_i^{(2)}$ for all the honest voter $\mathsf{V}_i \in \mathbb{V}$.

▶ Claim 9. $\mathcal{H}_3$ and $\mathcal{H}_2$ are indistinguishable.

Proof. The advantage of the adversary is bounded by the ZK property of $\mathsf{NIZK}_{\mathcal{R}_2}$ as defined by Def. 3. ◀

Hybrid $\mathcal{H}_4$: $\mathcal{H}_4$ is the same as $\mathcal{H}_3$ except the followings. During the vote phase, the simulator posts $c_i \leftarrow \mathsf{TFHE.Enc}(\mathtt{pk}, 0)$ for all the honest voter $\mathsf{V}_i \in \mathbb{V}$.

▶ Claim 10. $\mathcal{H}_4$ and $\mathcal{H}_3$ are indistinguishable.

Proof. The probability that any adversary $\mathcal{A}$ can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ is bounded by $\mathsf{AdvCPA}_{\mathcal{A}}(1^\lambda)$ and ciphertext transformative indistinguishability. More specifically, we now show the if there exists an adversary $\mathcal{A}$ who can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$, then we can construction an adversary $\mathcal{B}$ that can break the IND-CPA game of the underlying TFHE by reduction. During the IND-CPA game, $\mathcal{B}$ receives a public key $\mathtt{pk}^*$ from the challenger. There must be at least one honest trustee in this case, and with our loss of generality, assume T_x is honest. During the preparation phase, $\mathcal{B}$ posts $\mathtt{pk}^*$ as T_x's public key together with simulated proof. During the ballot casting phase, for each honest voter V_i, $i \in [n]$, $\mathcal{B}$ sends $m_0 := 0$ and $m_1 := s_i$ to the IND-CPA challenger, and receives c^*. $\mathcal{B}$ then computes $c' \leftarrow \mathsf{TFHE.Trans}(c^*, \{\mathtt{sk}_i\}_{i\in[k]\setminus\{x\}})$. It posts c' as the honest voter's encrypted ballot. It is easy to see that, when c^* encrypts m_0, the adversary's view is indistinguishable from $\mathcal{H}_4$; when c^* encrypts m_1, the adversary's view is indistinguishable from $\mathcal{H}_3$. Hence, if $\mathcal{A}$ can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ with non-negligible probability, then $\mathcal{B}$ can break the IND-CPA game with the same probability.

◀

The adversary's view of $\mathcal{H}_4$ is identical to the simulated view $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\mathrm{BB}}}_{\mathcal{F}_{\mathrm{SV}},\mathcal{S},\mathcal{Z}}$. Therefore, no PPT $\mathcal{Z}$ can distinguish the view of the ideal execution from the view of the real execution with more than negligible probability.

Case 2: $0 \le |\mathbb{V}_{\mathsf{corrupt}}| < n \ \wedge\ |\mathbb{T}_{\mathsf{corrupt}}| = k$.

Simulator. Similar as Case 1, the simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates honest voters $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, honest trustees $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$ and functionalities $\widehat{\mathcal{F}}_{\mathrm{CERT}}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- In the preparation phase:
  - Monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$, when a valid $(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, use $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Ext}^{\mathrm{RO}}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corresponding secret key $\overline{\mathtt{sk}}_j$.
- In the ballot casting phase:
  - Upon receiving $(\textsc{Leak}, \mathsf{sid}, \mathsf{V}_i, s_i)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest voter $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as V_i, following the protocol $\Pi_{\text{FHE-SV}}$ as if V_i receives $(\textsc{Cast}, \mathsf{sid}, s_i)$ from $\mathcal{Z}$.
  - Monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$, when a valid $(c_i, \pi_i^{(2)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted voter $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, uses the extracted $\left\{\overline{\mathtt{sk}}_j\right\}_{j\in[k]}$ to decrypt c_i to (V_i, s_i). The simulator $\mathcal{S}$ then acts as V_i to send $(\textsc{Cast}, \mathsf{sid}, s_i)$ to $\mathcal{F}_{\mathrm{SV}}$.

- In the tally phase:
  - The simulator $\mathcal{S}$ monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$; once a $\overline{\tau}_j, \pi_j^{(3)}$ is posted from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j to send $(\textsc{Tally}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{SV}}$.

Indistinguishability. The indistinguishability in this case is straightforward, as $\mathcal{S}$ never simulate a single message to either any corrupted parties or the external $\bar{\mathcal{G}}_{\mathrm{BB}}$. The simulator $\mathcal{S}$ knows all the honest voters' ballot from the external $\mathcal{F}_{\mathrm{SV}}$, it simply acts as the honest voters according to the protocol $\Pi_{\text{FHE-SV}}$. Meanwhile, it also extracts the ballot of the malicious voters by using the extracted trustees' secret keys. Hence, the simulator $\mathcal{S}$ can submit the extracted ballot to the external $\mathcal{F}_{\mathrm{SV}}$ on the malicious voters' behave. Therefore, when NIZK extraction for trustees' secret keys are successful, the view of $\mathcal{Z}$ in the ideal execution has identical distribution to the view of $\mathcal{Z}$ in the real execution.

Case 3: $|\mathbb{V}_{\mathsf{corrupt}}| = n \wedge 0 \leq |\mathbb{T}_{\mathsf{corrupt}}| \leq k$.

Simulator. Trivial case. There is nothing needs to extract, as the trustees do not have input. The simulator $\mathcal{S}$ just run trustee according to protocol $\Pi_{\text{FHE-SV}}$.

Indistinguishability. The view of $\mathcal{Z}$ in the ideal execution has identical distribution to the view of of $\mathcal{Z}$ in the real execution.

◀

B.4 Instantiation of TFHE via GSW

In this subsection, we present our construction TFHE. Assume there exist N players in our system, and each player has a $(\mathtt{pk}, \mathtt{sk})$ pair. Without lose of generality, for player i with $(\mathtt{pk}_i, \mathtt{sk}_i)$ generated from Keygen for $i \in [N]$.

- $\mathsf{param} \leftarrow \mathsf{TFHE.Setup}(1^\lambda)$: The algorithm takes as input the security parameter λ then outputs $\mathsf{param} := (n, m, q, \chi)$ as public parameter;
- $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow \mathsf{TFHE.Keygen}(1^\lambda)$: The algorithm first samples a public matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}$, a secret vector $\mathbf{s}_i \leftarrow \mathbb{Z}_q^{1\times n}$, and an error vector $\mathbf{e}_i \leftarrow \chi^{1\times m}$; Then, the algorithm computes $\mathbf{b}_i := \mathbf{s}_i \cdot \mathbf{B} + \mathbf{e}_i \pmod q \in \mathbb{Z}_q^{1\times m}$; The algorithm constructs and broadcasts the public key

  $$\mathtt{pk} := \mathbf{A} = \begin{pmatrix} \mathbf{B} \\ \mathbf{b} \end{pmatrix} \in \mathbb{Z}_q^{(n+1)\times m}$$

  and keeps secret key $\mathtt{sk} := \mathbf{t} := [-\mathbf{s}, 1] \in \mathbb{Z}_q^{1\times(n+1)}$ privately; Observe that

  $$[-\mathbf{s}|1] \cdot \begin{pmatrix} \mathbf{B} \\ \mathbf{b} \end{pmatrix} = \mathbf{e} \pmod q;$$

- $\mathtt{pk} := \mathsf{TFHE.CombinePK}(\mathtt{pk}_1, \ldots, \mathtt{pk}_N)$: It takes input as a set of public keys $(\mathtt{pk}_1, \ldots, \mathtt{pk}_N)$, and outputs a combined public key $\mathtt{pk}$. i.e., $\mathtt{pk} = \sum_{i=1}^N \mathtt{pk}_i$
- $c \leftarrow \mathsf{TFHE.Enc}(\mathtt{pk}, m)$: In this setting, each player uses the combine public key to encrypt his message m, in more detail:
  1. Most importantly, each player will broadcasts their public key $\mathtt{pk}_i$, and receives the other player's public keys, then generates a combine public key via $\mathsf{CombinePK}(\mathtt{pk}_1, \ldots, \mathtt{pk}_N)$:

  $$\mathtt{pk} := \sum_i^N \mathtt{pk}_i = \begin{pmatrix} \mathbf{B}^* \\ \mathbf{b}^* \end{pmatrix} \in \mathbb{Z}_q^{(n+1)\times m}$$

where $\mathbf{b}^* := \sum_i^N \mathbf{b}_i$ and $\mathbf{B}^* := \sum_i^N \mathbf{B}_i$;

2. Samples a random matrix $\mathbf{R}_i \leftarrow \{0,1\}^{m\times(n+1)\ell}$, then, computes and broadcasts

$$\mathbf{C} := \begin{pmatrix}\mathbf{B}^* \\ \mathbf{b}^*\end{pmatrix} \cdot \mathbf{R}_i + m_i \cdot \mathbf{G} \pmod q \in \mathbb{Z}_q^{(n+1)\times(n+1)\ell}$$

Here, we stress that, in general encryption scheme, each player encrypts $\mathsf{msg} \in \{0,1\}$ under his public key $\mathtt{pk}$ by using $\mathbf{C}_i \leftarrow \mathsf{Enc}(\mathtt{pk}_i, m_i) := \mathtt{pk}_i\mathbf{R}_i + m_i\mathbf{G}$ where the $\mathtt{pk}_i$ is not the common public key.

- $c^* \leftarrow \mathsf{TFHE.Eval}(C, \mathtt{pk}, c_1, \cdots, c_N)$: **Homomorphic evaluation algorithm,** upon receiving all the encrypted data from other players, each player invokes the Eval algorithm (e.g., addition or multiplication) to generate the evaluation ciphertext. We stress that the Eval algorithm is the same as the evaluation algorithm of Gentry et al. [29].
- $\mathtt{sk} \leftarrow \mathsf{CombineSK}(\mathtt{sk}_1, \ldots, \mathtt{sk}_N)$. It takes input as a set of secret key $(\mathtt{sk}_1, \ldots, \mathtt{sk}_k)$, and outputs combined secret key $\mathtt{sk}$. More concretely, each player uses the secret key share functional polynomial f_i to share the secret key share, e.g., $f(\mathbf{x}) = \mathtt{sk} + r_1\mathbf{x}^1 + r_2\mathbf{x}^2 + \cdots + r_N\mathbf{x}^N$ and sends each f_j to the player j for $j \in [N]$. Then, the player i re-constructs these shares to generate $\mathtt{sk}_i := \sum_{j\in S} f_j(i)$. For example, we parse $\mathtt{sk}_i$ into k pieces, $\mathtt{sk}_i := (f_i(1), \cdots, f_i(N))$, at the end of secret share, we set $\mathtt{sk}_i := (f_1(i), \cdots, f_N(i))$.
- $y \leftarrow \mathsf{TFHE.Dec}(c^*, \mathtt{sk}_1, \cdots, \mathtt{sk}_N)$:
  1. Upon receiving the shares from other players, the player i combines his shares of secret key by computing $\mathtt{sk}_i := \sum_{j\in S} f_j(i)$ and broadcasts $\mu_i := (\sum_{j\in S} \mathtt{sk}_j \cdot c^*) \cdot \mathbf{G}^{-1}(\mathbf{w}^T) + smdg_i$;
  2. Upon receiving all the partial messages $\{\mu_i\}_{i\in T}$, each player picks an arbitrary subset $T \subseteq S \subseteq [N]$ such that $|T| = [N/2] + 1$. Then, they use the "*Lagrange interpolation*" polynomial to compute $result = \sum_{k\in T} \delta_k(0) \cdot \mu_k = \lfloor \frac{q}{2} \rfloor \cdot m + noise$ for $k \in T$;
  3. Finally, they output m.
- $\mu_i \leftarrow \mathsf{TFHE.ShareDec}(\mathtt{sk}_i, c^*)$. It takes the secret key of player i and the evaluated ciphertext as input. Upon receiving the shares from other players, the player i combines his shares of secret key by computing $\mathtt{sk}_i := \sum_{j\in S} f_j(i)$ and broadcasts the partial message $\mu_i := (\sum_{j\in S} \mathtt{sk}_j \cdot c^*) \cdot \mathbf{G}^{-1}(\mathbf{w}^T) + \mathsf{smdg}_i$;
- $m \leftarrow \mathsf{TFHE.ShareCombine}(c, \mu_1, \ldots, \mu_k)$. It takes input as a ciphertext c and k decryption shares $(\mu_1, \ldots, \mu_k)$, and outputs a plaintext m. More concretely, upon receiving all the partial messages $\{\mu_i\}_{i\in T}$, each player picks an arbitrary subset $T \subseteq S \subseteq [N]$ such that $|T| = [N/2] + 1$. Then, they use the "*Lagrange interpolation*" polynomial to compute $result = \sum_{k\in T} \delta_k(0) \cdot \mu_k = \lfloor \frac{q}{2} \rfloor \cdot m + noise$ for $k \in T$; Lastly, outputs m.

▶ **Theorem 11.** *The construction* TFHE *above is a secure publicly evaluable key-homomorphic threshold FHE under the LWE assumption.*

Proof. To prove the above theorem, we need to show

1). Correct Key Combination: Consider the combination of keys, it is easily seen that

$$\begin{aligned} \mathtt{pk}^* &= \mathtt{pk}_1 + \mathtt{pk}_2 + \cdots + \mathtt{pk}_N = \sum_{i=1}^N \mathbf{b}_i = \sum_{i=1}^N \big(\mathbf{s}_i \cdot \mathbf{B} + \mathbf{e}_i \pmod q \in \mathbb{Z}_q^{1\times m}\mathbf{s}_i\big) \cdot \mathbf{B} \\ &= (\sum_{i=1}^N \mathbf{s}_i)\mathbf{B} + (\sum_{i=1}^N \mathbf{e}_i) \pmod q. \end{aligned}$$

Obliviously, then $(\mathtt{pk}^*, \mathtt{sk}^*)$ are valid key tuples.

2). Ciphertext transformative indistinguishability: We note that, the PPT algorithm Trans takes input as the current ciphertext c under the set $\{\mathtt{sk}_i\}$ for $i \in [k]$, and outputs the transformed ciphertext $c' \approx c$. As mentioned earlier, in our setting, we obtain that, $\mathbf{C}' := \mathsf{Trans}(\mathbf{C}, \{\mathtt{sk}_i\}_{i\in[k]}) = \begin{pmatrix} \mathbf{B} \\ \hline \sum_{i\in j} \mathbf{b}_i \end{pmatrix} \cdot \bar{\mathbf{R}} + m \cdot \mathbf{G}$, where we recall the original ciphertext as follows $\mathbf{C} = \begin{pmatrix} \mathbf{B} \\ \hline \mathbf{b} \end{pmatrix} \cdot \mathbf{R} + m \cdot \mathbf{G}$ under the secret key $\mathtt{sk} := \mathbf{t} := [-\mathbf{s}, 1] \in \mathbb{Z}_q^{1\times(n+1)}$. Notably $\mathbf{b} = \mathbf{s} \cdot \mathbf{B} + \mathbf{e} \pmod q$. In order to prove the $\mathbf{C}'$ and $\mathbf{C}$ indistinguishability, we only consider $\begin{pmatrix} \mathbf{B} \\ \hline \sum_{i\in j} \mathbf{b}_i \end{pmatrix} \cdot \bar{\mathbf{R}}$ and $\begin{pmatrix} \mathbf{B} \\ \hline \mathbf{b} \end{pmatrix} \cdot \mathbf{R}$ indistinguishability. In a simple, the simulator can easily obtain the original ciphertext and the public keys which from the parties. Once the simulator obtain the randomness from one of the parties, he could create a matrix $\bar{\mathbf{R}} = \mathbf{R} + \mathbf{Y} \in \{0,1\}^{m\times(n+1)\ell}$ for $\mathbf{z} = \sum_{i\in[k]\setminus[j]} \mathbf{b}_i \mathbf{R} + (\sum_{i\in[k]\setminus[j]} \mathbf{s}_i \cdot \mathbf{B} + \sum_{i\in[k]\setminus[j]} \mathbf{e}_i)\mathbf{Y} \in \mathbb{Z}_q^{1\times m}$. Hence, they are identical and there is no PPT adversary can distinguish them.

3). Share-simulation indistinguishability: We first define the $\mathsf{SimShareDec}(c, m, \{\mu_i\}_{i\in\mathcal{I}})$, then fix $j^* \in \bar{T} = [N] \setminus T$. Sample the partial message μ_j uniformly and let $\mu_{j^*} := \begin{pmatrix} \mathbf{B}^* \\ \hline \mathbf{b}^* \end{pmatrix} \cdot \mathbf{R}_i - \sum_{i\in N, i\neq j^*} \mu_i$ and output $\{\mu_j\}_{j\in\bar{T}}$. By correct share decryption, we know that regardless of how $\{\mu_j\}_{j\in\bar{T}}$ were created, $\mu_{j^*} := \begin{pmatrix} \mathbf{B}^* \\ \hline \mathbf{b}^* \end{pmatrix} \cdot \mathbf{R}_i - \sum_{i\in N, i\neq j^*} \mu_i$. Since this is a deterministic function of the rest of the variables, we simply need to prove that $\{\mu_j := (\mathtt{sk}_j^* \cdot c^*) \cdot \mathbf{G}^{-1}(\mathbf{w}^T) + \mathsf{smdg}_j\}_{\{j\in\bar{T}, j\neq j^*\}} \approx_c \{\mu_j \leftarrow \mathbb{Z}_q\}_{\{j\in\bar{T}, j\neq j^*\}}$. Obliviously, inspired by the security of TFHE, utilizing the leftover hash lemma and the LWE assumption, we can prove the above equation satisfy the property of computational indistinguishability. ◀

B.5 Fully homomorphic encryption

A fully homomorphic encryption scheme FHE consists of a tuple of algorithms: $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ as follows.

- $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$. The algorithm Setup takes input as the security parameter λ, and outputs public parameters param. All the other algorithms implicitly take param as input.
- $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{Keygen}(\mathsf{param})$. The algorithm Keygen takes input as the public parameter param, and outputs a public key $\mathtt{pk}$, a secret key $\mathtt{sk}$.
- $c \leftarrow \mathsf{Enc}(\mathtt{pk}, m)$. The algorithm Enc takes input as the public key $\mathtt{pk}$ and the message m, and outputs the ciphertext c.
- $c' := \mathsf{Eval}(\mathtt{pk}, \mathcal{F}, c_1, \ldots, c_n)$. The algorithm Eval takes input as the public (a.k.a., evaluation) key $\mathtt{pk}$, the description of the evaluation function (circuit) $\mathcal{F}$, and a set of ciphertexts $c_1, \ldots, c_n$, and outputs the result ciphertext c'.
- $m \leftarrow \mathsf{Dec}(\mathtt{sk}, c)$. The algorithm Dec takes input as the secret key $\mathtt{sk}$ and a ciphertext c, and outputs the decrypted plaintext m.

▶ **Definition 12.** We say $\mathsf{FHE} = \{\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}\}$ is a *secure* fully homomorphic encryption if the following properties hold:

- Correctness: The correctness properties are required as follows:

- For any λ, $m \in \{0,1\}^*$, and $(\mathtt{pk}, \mathtt{sk})$ output by $\mathsf{Keygen}(1^\lambda)$, we have that

$$m = \mathsf{Dec}\Big(\mathtt{sk}, \big(\mathsf{Enc}(\mathtt{pk}, m)\big)\Big);$$

- For any λ, any $m_1, \cdots, m_l \in \{0,1\}^*$, and $C \in \mathcal{C}_\lambda$, we have that

$$\mathcal{C}(m_1, \cdots, m_\ell) \quad = \quad \mathsf{Dec}\Big(\mathtt{sk}, \big(\mathsf{Eval}\big(\mathtt{pk}, (C, \mathsf{Enc}(\mathtt{pk}, m_1), \cdots, \mathsf{Enc}(\mathtt{pk}, m_\ell))\big)\big)\Big).$$

- IND-CPA security: We say that a FHE scheme achieves *indistinguishability under plaintext attacks (IND-CPA)* if for any PPT adversary $\mathcal{A}$ the following advantage AdvCPA is negligible.

  $\underline{\text{EXPERIMENT}^{\mathsf{CPA}}(1^\lambda)}$

  1. Run $\mathsf{param} \leftarrow \mathsf{FHE.Setup}(1^\lambda)$.
  2. Run $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{FHE.Keygen}(\mathsf{param})$;
  4. $\mathcal{A}(\mathtt{pk})$ outputs m_0, m_1 of equal length;
  5. Pick $b \leftarrow \big\{0,1\big\}$; Run $c \leftarrow \mathsf{FHE.Enc}(\mathtt{pk}, m_b)$;
  6. $\mathcal{A}(c)$ outputs b^*; It returns 1 if $b = b^*$; else, returns 0.

  We define the advantage of $\mathcal{A}$ as

$$\mathsf{AdvCPA}_{\mathcal{A}}(1^\lambda) = \left|\Pr[\text{EXPERIMENT}^{\mathsf{CPA}}(1^\lambda) = 1] - \frac{1}{2}\right| .$$

B.6 Gentry-Sahai-Waters (GSW) construction

Let k be a security parameter and let L be the number of levels for the somewhat homomorphic scheme. We describe the algorithms that form the GSW scheme [29]. The algorithm is originally defined in terms of the functions $\mathsf{BitDecomp}, \mathsf{BitDecomp}^{-1}$ and $\mathsf{Flatten}$, but we tend to follow the formulation in [3, 41] and so use the matrix $\mathbf{G}$.

- $\mathsf{GSW.Setup}(1^k, 1^L)$:
  1. Choose a modulus q of $\kappa = \kappa(k, L)$ bits, parameter $n = n(k, L) \in \mathbb{N}$, and error distribution $\chi = \chi(k, L)$ on $\mathcal{Z}$ so that the (n, q, m, χ)-LWE problem achieves at least 2^k security against known attacks.
     Choose a parameter $m = m(k, L) = O(n \log(q))$;
  2. Output: $\mathsf{param} = (n, q, m, \chi)$.
     We also use the notation $\ell = \lfloor \log(q) \rfloor + 1$ and $N = (n+1) \cdot \ell$.
- $\mathsf{GSW.Keygen}(\mathsf{param})$:
  1. Sample uniformly $\mathbf{t} = (t_1, \ldots, t_n)^T \leftarrow \mathbb{Z}_q^n$ and compute

$$\mathbf{s} \leftarrow (1, -\mathbf{t}^T)^T = (1, -t_1, \cdots, -t_n)^T \in \mathbb{Z}_q^{(n+1)\times 1};$$

  2. Generate a matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{m\times n}$ uniformly and a vector $\mathbf{e} \leftarrow \chi^m$;
  3. Compute $\mathbf{b} = \mathbf{Bt} + \mathbf{e} \in \mathbb{Z}_q^m$ and construct the matrix $\mathbf{A} = (\mathbf{b}|\mathbf{B}) \in \mathbb{Z}_q^{m\times(n+1)}$ as the vector $\mathbf{b}$ followed by the n columns of $\mathbf{B}$.
     Observe that

$$\mathbf{As} = (\mathbf{b}|\mathbf{B})\mathbf{s} = (\mathbf{Bt} + \mathbf{e}|\mathbf{B}) \begin{pmatrix} 1 \\ -\mathbf{t} \end{pmatrix} = \mathbf{Bt} + \mathbf{e} - \mathbf{Bt} = \mathbf{e}.$$

  4. Return $\mathtt{sk} \leftarrow \mathbf{s}$ and $\mathtt{pk} \leftarrow \mathbf{A}$.

- $\mathbf{C} \leftarrow \mathsf{GSW.Enc}(\mathsf{param}, \mathtt{pk}, \mu)$: In order to encrypt one-bit messages $\mu \in \{0,1\}$:
  1. Let $\mathbf{G}$ be the $(n+1) \times N$ gadget matrix as above;
  2. Sample uniformly a matrix $\mathbf{R} \leftarrow \{0,1\}^{m \times N}$;
  3. Compute $\mathbf{C} = \mu\mathbf{G} + \mathbf{A}^T\mathbf{R} \pmod q \in \mathbb{Z}_q^{(n+1)\times N}$.
- $\mu' \leftarrow \mathsf{GSW.Dec}(\mathsf{param}, \mathtt{sk}, \mathbf{C})$:
  1. We have $\mathtt{sk} = \mathbf{s} \in \mathbb{Z}_q^{n+1}$;
  2. Define a vector $\mathbf{w} = [\lceil q/2 \rceil | 0, \cdots, 0] \in \mathbb{Z}_q^{1\times(n+1)}$
  3. Compute $v = \mathbf{s}^T\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{w}^T) \in \mathbb{Z}_q$ and output $\mu = || \lfloor \frac{v}{q/2} \rceil ||$ as the decrypted message. So if $|\mu| \leq q/4$ then return 0 otherwise return 1.
- $\mathsf{GSW.Eval}(\mathsf{param}, \mathbf{C}_1, \cdots, \mathbf{C}_\ell)$:
  - $\mathsf{GSW.Add}(\mathbf{C}_1, \mathbf{C}_2)$: output

$$\mathbf{C}_1 + \mathbf{C}_2 = (\mu_1 + \mu_2)\mathbf{G} + \mathbf{A}^T(\mathbf{R}_1 + \mathbf{R}_2) \in \mathbb{Z}_q^{(n+1)\times N};$$

  - $\mathsf{GSW.Mult}(\mathbf{C}_1, \mathbf{C}_2)$: Compute $\mathbf{G}^{-1}(\mathbf{C}_2) \in \{0,1\}^{N\times N}$ and output $\mathbf{C}_1\mathbf{G}^{-1}(\mathbf{C}_2)$. Note that

$$\begin{aligned}
\mathbf{C}_1\mathbf{G}^{-1}(\mathbf{C}_2) &= \big(\mu_1\mathbf{G} + \mathbf{A}^T\mathbf{R}_1\big)\mathbf{G}^{-1}(\mathbf{C}_2) \\
&= \mu_1\mathbf{C}_2 + \mathbf{A}^T\mathbf{R}_1\mathbf{G}^{-1}(\mathbf{C}_2) \\
&= \mu_1\mu_2\mathbf{G} + \mathbf{A}^T\mathbf{R}_1\mathbf{G}^{-1}(\mathbf{C}_2) + \mu_1\mathbf{A}^T\mathbf{R}_2 \\
&= \mu_1\mu_2\mathbf{G} + \mathbf{A}^T\big(\mathbf{R}_1\mathbf{G}^{-1}(\mathbf{C}_2) + \mu_1\mathbf{R}_2\big) \in \mathbb{Z}_q^{(n+1)\times N}.
\end{aligned}$$

  One may also compute a homomorphic NAND gate by outputting $\mathbf{G} - \mathbf{C}_1\mathbf{G}^{-1}(\mathbf{C}_2)$.

▶ Remark. Note that the formulation of the decryption algorithm in [41] is to choose an appropriate vector $\mathbf{w}$ and compute $\mathbf{sCG}^{-1}(\mathbf{w}^T)$. This is considerably less efficient than the original GSW decryption algorithm (both in terms of computation time and also the size of the error term). Hence we employ the original GSW decryption algorithm for our scheme.

There is also a variant of the scheme that handles messages in $\mathbb{Z}_q$ when q is a power of two. We refer to [29] for the details.

B.6.1 Security

A sketch proof is given in [29] of the following theorem.

▶ **Theorem 13.** *Let (n, q, m, χ) be such that the $\mathsf{LWE}_{(n,q,m,\chi)}$ assumption holds and let $m = O(n \log(q))$. Then the GSW scheme is IND-CPA secure.*

The main step in the proof is showing that $(\mathbf{A}, \mathbf{RA})$ is computationally indistinguishable from uniform.

▶ **Definition 14** ([29, 3, 14, 13])**.** If the ciphertexts $\mathbf{C} = \mu\mathbf{G} + \mathbf{A} \cdot \mathbf{R}$, along with the secret key $\mathbf{s} = (-\mathbf{t}, 1)$, then the noise of $\mathbf{C}$ is the infinity norm of the noise vector: $\mathsf{noise}_{(\mathbf{s},\mu)}(\mathbf{C}) = \|\mathbf{C} - \mu\mathbf{G}\|_\infty$, i.e., $\mathsf{noise}_{(\mathbf{s},\mu)}(\mathbf{C}) = \|\mathbf{tA} \cdot \mathbf{R}\| = \|\mathbf{e} \cdot \mathbf{R}\| \leq mB \cdot m \leq E$.

▶ **Lemma 15** ([29, 3, 14, 13])**.** *For the ciphertexts $\mathbf{C} = \mu\mathbf{G} + \mathbf{A} \cdot \mathbf{R} \in \mathbb{Z}_q^{n\times m}$, along with the secret key $\mathbf{s} = (-\mathbf{t}, 1)$ and $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$, then the noise in negation, addition and multiplication is bounded as follows:*

- ***Addition:*** *for all messages* μ_1, $\mu_2 \in \{0,1\}$, *it holds that* $\mathsf{noise}_{(\mathbf{s},\mu_1+\mu_2)}(\mathbf{C}_1 + \mathbf{C}_2) \leq \mathsf{noise}_{(\mathbf{s},\mu_1)}(\mathbf{C}_1) + \mathsf{noise}_{(\mathbf{s},\mu_2)}(\mathbf{C}_2)$*;*
- ***Multiplication:*** *for all messages* μ_1, $\mu_2 \in \{0,1\}$, *it holds that* $\mathsf{noise}_{(\mathbf{s},\mu_1\mu_2)}(\mathbf{C}_1\mathbf{G}^{-1}(\mathbf{C}_2)) \leq \mu_1 \cdot \mathsf{noise}_{(\mathbf{s},\mu_2)}(\mathbf{C}_2) + m \cdot \mathsf{noise}_{(\mathbf{s},\mu_1)}(\mathbf{C}_1)$ *for an efficiently computable function* $\mathbf{G}^{-1} : \mathbb{Z}_q^n \to \mathbb{Z}_q^m$*. i.e.,* $\mathsf{noise}_{(\mathbf{s},\mu_1\mu_2)}(\mathbf{C}_1\mathbf{G}^{-1}(\mathbf{C}_2)) \leq \|\mu_1 \cdot (\mathbf{e}_2\mathbf{R}_2) + (\mathbf{e}_1\mathbf{R}_1) \cdot \mathbf{G}^{-1}(\mathbf{C}_2)\|$.
- ***Negation:*** *for all message* $\mu \in \{0,1\}$, *it holds that* $\mathsf{noise}_{(\mathbf{s},1-\mu)}(\mathbf{G} - \mathbf{C}) = \mathsf{noise}_{(\mathbf{s},\mu)}(\mathbf{C})$.

B.7 LWE assumption

▶ **Definition 16** ([12] Def2.1)**.** A distribution ensemble $\chi = \chi(\lambda)$ over the integers is called B-bounded (denoted $|\chi| \leq B$) if there exists:

$$\Pr_{x \xleftarrow{\$} \chi}[|x| \geq B] \leq 2^{-\tilde{\Omega}(n)}$$

▶ **Definition 17** (LWE Distribution)**.** For the security parameter λ, let $n = n(\lambda)$ and $m = m(\lambda)$ be integers, let $\chi = \chi(\lambda)$ be error distribution over $\mathcal{Z}$ bounded by $B = B(\lambda)$, and let $q = q(\lambda) \geq 2$ be an integer modulus for any polynomial $p = p(\lambda)$ such that $q \geq 2^p \cdot B$. Then, sample a vector $\mathbf{s} \in \mathbb{Z}_q^{n\times 1}$ called the secret, the LWE distribution $\mathcal{A}_{\mathbf{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ uniformly at random, choosing $\mathbf{e} \leftarrow \chi^{m\times 1}$, and outputting $\big(\mathbf{A}, \mathbf{b} = \mathbf{A}\cdot\mathbf{s} + \mathbf{e} \pmod q\big)$.

We define the decisional version as follows,

▶ **Definition 18** (Decision-LWE$_{n,q,\chi,m}$)**.** Assume given an independent sample $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^{m\times 1}$, where the sample is distributed according to either: (1) $\mathcal{A}_{\mathbf{s},\chi}$ for a uniformly random $\mathbf{s} \in \mathbb{Z}_q^n$ (i.e., $\{(\mathbf{A},\mathbf{b}) : \mathbf{A} \leftarrow \mathbb{Z}_q^{m\times n}, \mathbf{s} \leftarrow \mathbb{Z}_q^{n\times 1}, \mathbf{e} \leftarrow \chi^{m\times 1}, \mathbf{b} = \mathbf{A}\cdot\mathbf{s} + \mathbf{e} \pmod q\}$), or (2) the uniform distribution (i.e., $\{(\mathbf{A},\mathbf{b}) : \mathbf{A} \leftarrow \mathbb{Z}_q^{m\times n}, \mathbf{b} \leftarrow \mathbb{Z}_q^{m\times 1}\}$). Then, the above two distributions are computationally indistinguishable.

▶ Remark. Regev and others [45, 42, 43, 41] show that reductions between the LWE assumption and approximating the shortest vector problem in lattices (for appropriate parameters). We omit the corollary of these schemes' results. More details will be find [45, 42, 43, 41].

▶ **Lemma 19** (Smudging Lemma)**.** *Let* $B_1 = B_1(\lambda)$*, and* $B_2 = B_2(\lambda)$ *be positive integers and let* $v^{\mathsf{sm}} \in [-B_1, B_1]$ *be a fixed integer. Let* $v_2^{\mathsf{sm}} \leftarrow [-B_2, B_2]$ *be chosen uniformly at random. Then the distribution of* v_2^{sm} *is statistically indistinguishable from that of* $v_2^{\mathsf{sm}} + v_1^{\mathsf{sm}}$ *as long as* $B_1/B_2 = \mathsf{negl}(\lambda)$*.*

▶ **Lemma 20** ([38])**.** *For any* $N \geq m\lceil\log q\rceil$ *there exists a computable gadget matrix* $\mathbf{G} \in \mathbb{Z}_q^{m\times N}$ *and an efficiently computable deterministic inverse (a.k.a., "short preimage") function* $\mathbf{G}^{-1}(\cdot)$*. The inverse function* $\mathbf{G}^{-1}(\mathbf{M})$ *takes as input a matrix* $\mathbf{M} \in \mathbb{Z}_q^{m\times m'}$ *for any* m' *and outputs a matrix* $\mathbf{G}^{-1}(\mathbf{M}) \in \{0,1\}^{N\times m'}$ *such that* $\mathbf{G}\mathbf{G}^{-1}(\mathbf{M}) = \mathbf{M}$*.*

C MPC based construction

The presented TFHE-based construction is used to illustrate the core idea. In practice, FHE schemes may still be hundreds times slower than the state-of-the-art MPC protocols, especially when NIZK proofs are involved. In fact, the construction described in Section B can be viewed as a special case of an MPC protocol in the server-client setting, where the trustees $\mathbb{T}$ form the MPC players. The voters submit their statements, and the trustees then jointly evaluate the TallyProcess circuit.

In the following, we show how to eliminate the needs of a FHE using so-called publicly auditable MPC. Note that the main difference between a conventional MPC protocol and an e-voting system is that the e-voting system should still ensure the integrity of an election process even when all the trustees are corrupted. Whereas, a conventional MPC protocol does not ensure computation correctness when all the players are corrupted.

Baum et al. [6], proposed a publicly auditable MPC in the $\bar{\mathcal{G}}_{\mathrm{BB}}$-hybrid model. Their scheme is based on SPDZ [27, 26], but it can be extended to support most other later SPDZ variants along this line of research. The general idea to make an MPC system publicly auditable is to attach each shared value with a (Pedersen) commitment so that the same linear/opening operations of the shared value can be carried out on the corresponding commitments. Those commitments are posted on the $\bar{\mathcal{G}}_{\mathrm{BB}}$, so that everyone can perform the same MPC online phase circuit on the commitments and check if the opening of the resulting commitment is consistent with the MPC output. Hereby, due to space limitation, we will omit the MPC construction and refer interested readers to [6] for details. In the following, we first give another building block.

C.1 Protocol description

In this section, we formally describe our MPC-based construction for statement voting. we assume there exists an MPC protocol Π_{MPC} that UC-realize $\mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}$, where $\mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}$ is the MPC functionality (as described in Fig. 1 of [6]) and $\mathcal{C}$ is the statement voting circuit depicted in Fig. 13, below. $\mathcal{C}$ takes public input as each trustee T_i's partial public key $\overline{\mathtt{pk}}_i$, and a set of encrypted ballots $\{c_j \leftarrow \mathsf{TE.Enc}(\mathtt{pk}, (\mathsf{V}_i, s_i))\}_{j\in[n]}$, where s_i is the voter V_i's statement. Meanwhile, $\mathcal{C}$ also takes private inputs as a random coin α_j from each trustee T_j, $j \in [k]$. $\mathcal{C}$ first uses α_j to generate $(\hat{\mathtt{pk}}_j, \hat{\mathtt{sk}}_j) \leftarrow \mathsf{TE.Keygen}(\mathsf{param}; \alpha_j)$; it then checks if $\hat{\mathtt{pk}}_j = \overline{\mathtt{pk}}_j$, $j \in [k]$. If verified, $\mathcal{C}$ uses $\{\hat{\mathtt{sk}}_j\}_{j\in[k]}$ to decrypt the ciphertexts $\{c_i\}_{i\in[n]}$ to obtains the ballots $\{B_i\}_{i\in[n]}$. It then computes and outputs the tally $\tau \leftarrow \mathsf{TallyProcess}(B_1, \ldots, B_n)$.

Statement voting circuit

Public input: $\{\overline{\mathtt{pk}}_j\}_{j\in[k]}$ and $\{c_i\}_{i\in[n]}$
Private input: $\{\alpha_j\}_{j\in[k]}$
Public output: τ

- For $j \in [k]$, generate $(\hat{\mathtt{pk}}_j, \hat{\mathtt{sk}}_j) \leftarrow \mathsf{TE.Keygen}(\mathsf{param}; \alpha_j)$.
- Check if $\hat{\mathtt{pk}}_j = \overline{\mathtt{pk}}_j$, $j \in [k]$; otherwise, return $\perp$.
- Set $\mathtt{sk} := \mathsf{TE.CombineSK}(\hat{\mathtt{sk}}_1, \ldots, \hat{\mathtt{sk}}_k)$.
- For $i \in [n]$, compute $B_i \leftarrow \mathsf{TE.Dec}(\mathtt{sk}, c_i)$.
- Return $\tau \leftarrow \mathsf{TallyProcess}(B_1, \ldots, B_n)$.

Figure 13 Statement voting circuit $\mathcal{C}$

The protocol is designed in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\mathrm{CERT}}, \mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}\}$-hybrid world and it consists of three phases: preparation, ballot casting, and tally. Again, for the sake of notation simplicity, we omit the processes of filtering invalid messages on $\bar{\mathcal{G}}_{\mathrm{BB}}$. In practice, $\bar{\mathcal{G}}_{\mathrm{BB}}$ contains many messages with invalid signatures, and all those messages should be ignored. We assume all the parties implicitly have a common input $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$.

C.1.1 Preparation phase

As depicted in Fig. 14, the preparation phase is the same as the TFHE based construction, except it uses TE instead.

Preparation

Upon receiving $(\textsc{InitialTrustee}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the trustee T_j, $j \in [k]$, operates as the follows:

- Generate $(\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) \leftarrow \mathsf{TE.Keygen}(\mathsf{param}; \alpha_j)$ where α_j is the fresh randomness, and then compute

$$\pi_j^{(1)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{11}} \left\{ \ (\overline{\mathtt{pk}}_j), (\alpha_j, \overline{\mathtt{sk}}_j) : (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{TE.Keygen}(\mathsf{param}; \alpha_j) \ \right\}$$

- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and receives $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)} \rangle)$ to $\bar{\mathcal{G}}_{\text{BB}}$.

Figure 14 MPC based statement voting $\Pi_{\text{MPC-SV}}$ in $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}, \mathcal{F}_{\text{MPC}}^{\mathcal{C}}\}$-hybrid world (Part I).

C.1.2 Ballot casting phase

As depicted in Figure 15, the ballot casting phase is also the same as the TFHE scheme, except TE is used instead.

Ballot Casting

Upon receiving $(\textsc{Cast}, \mathsf{sid}, s_i)$ from the environment $\mathcal{Z}$, the voter V_i, $i \in [n]$ operates as the follows:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\text{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\text{BB}}$. If $\left\{ \langle \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)} \rangle \right\}_{j \in [k]}$ is contained in state, then for $j \in [k]$, send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), b_j^{(1)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; If $\prod_{j=1}^{k} b_j^{(1)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Verify}(\overline{\mathtt{pk}}_j, \pi_j^{(1)}) = 1$ for $j \in [k]$. If any of the checks is invalid, halt.
- Compute and store $\mathtt{pk} := \mathsf{TE.CombinePK}(\{\overline{\mathtt{pk}}_j\}_{j=1}^{k})$.
- Encrypt $c_i \leftarrow \mathsf{TE.Enc}(\mathtt{pk}, (\mathsf{V}_i, s_i); \beta_i)$ where β_i is the fresh randomness, and then compute

$$\pi_i^{(2)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{12}} \left\{ \ (\overline{\mathtt{pk}}, c_i), (\mathsf{V}_i, s_i, \beta_i) : c_i = \mathsf{TE.Enc}(\mathtt{pk}, (\mathsf{V}_i, s_i); \beta_i) \ \right\}$$

- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (c_i, \pi_i^{(2)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{V}_i, \mathsf{ssid}')$ for some ssid',
  and receive $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (c_i, \pi_i^{(2)}), \sigma_i^{(2)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (c_i, \pi_i^{(2)}), \sigma_i^{(2)} \rangle)$ to $\bar{\mathcal{G}}_{\text{BB}}$.

Figure 15 MPC based statement voting $\Pi_{\text{MPC-SV}}$ in $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}, \mathcal{F}_{\text{MPC}}^{\mathcal{C}}\}$-hybrid world (Part II).

C.1.3 Tally phase

The tally phase is depicted in Figure 16. It is the same as the FHE-based scheme except all the trustees invoke $\mathcal{F}_{\text{MPC}}^{\mathcal{C}}$ to calculate the tally circuit.

Tally

Upon receiving $(\textsc{Tally}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the trustee T_j, where $j \in [k]$, operates as the follows:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. If $\left\{\langle \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)} \rangle\right\}_{j \in [k]}$ is contained in state, then for $j \in [k]$, send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), b_j^{(1)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; If $\prod_{j=1}^{k} b_j^{(1)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Verify}(\overline{\mathtt{pk}}_j, \pi_j^{(1)}) = 1$ for $j \in [k]$. If any of the checks is invalid, halt.
- For $i \in [n]$, if $\langle \mathsf{ssid}, (c_i, \pi_i^{(2)}), \sigma_i^{(2)} \rangle$ is contained in state, then send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (c_i, \pi_i^{(2)}), \sigma_i^{(2)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (c_i, \pi_i^{(2)}), b_i^{(2)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; if $b_i^{(2)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\overline{\mathtt{pk}}, c_i), \pi_i^{(2)}) = 1$. If any of the above checks is invalid, reset $c_i := \perp$.
- Send $(\textsc{Input}, \mathsf{sid}, \alpha_j, \{\mathsf{pk}_\ell\}_{\ell \in [k]}, \{c_i\}_{i \in [n]})$ to $\mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}$, and obtain τ from $\mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}$.
- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, \tau)$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and receives $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, \tau), \sigma_j^{(3)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, \tau, \sigma_j^{(3)} \rangle)$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Upon receiving $(\textsc{ReadResult}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the voter V_i, $i \in [n]$ operates as the follows:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$.
- Fetch τ from state and return $(\textsc{ReadResultReturn}, \mathsf{sid}, \tau)$ to the environment $\mathcal{Z}$.

Figure 16 MPC based voting scheme $\Pi_{\text{MPC-SV}}$ in $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}, \mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}\}$-hybrid world (Part III).

C.2 Security

We have the following the theorem.

▶ **Theorem 21.** *Protocol $\Pi_{\text{MPC-SV}}$ described in Figure 14, Figure 15 and Figure 16 UC-realizes $\mathcal{F}_{\mathrm{SV}}$ in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}, \mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}\}$-hybrid world against static corruption.*

Proof. To prove the theorem, we construct a simulator $\mathcal{S}$ such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\mathsf{EXEC}_{\Pi_{\text{MPC-SV}}, \mathcal{A}, \mathcal{Z}}^{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}, \mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}}$ where the parties $\mathbb{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ and $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$ run protocol $\Pi_{\text{MPC-SV}}$ in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}, \mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}\}$-hybrid world and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathsf{EXEC}_{\mathcal{F}_{\mathrm{SV}}, \mathcal{S}, \mathcal{Z}}^{\bar{\mathcal{G}}_{\mathrm{BB}}}$ where the parties interact with functionality $\mathcal{F}_{\mathrm{SV}}$ in the $\bar{\mathcal{G}}_{\mathrm{BB}}$-hybrid model and corrupted parties are controlled by the simulator $\mathcal{S}$. Let $\mathbb{V}_{\mathsf{corrupt}} \subseteq \mathbb{V}$ and $\mathbb{T}_{\mathsf{corrupt}} \subseteq \mathbb{T}$ be the set of corrupted voters and trustees, respectively. We consider following cases.

Case 1: $0 \leq |\mathbb{V}_{\mathsf{corrupt}}| < n \;\wedge\; 0 \leq |\mathbb{T}_{\mathsf{corrupt}}| < k$.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates honest voters $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, honest trustees $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$ and functionalities $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and $\mathcal{F}_{\mathrm{MPC}}^{\mathcal{C}}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- In the preparation phase:

  - Upon receiving $(\textsc{InitialTrusteeNotify}, \mathsf{sid}, \mathsf{T}_j)$ from the external $\mathcal{F}_{\text{SV}}$ for an honest trustee $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j, following the protocol $\Pi_{\text{MPC-SV}}$ as if T_j receives $(\textsc{InitialTrustee}, \mathsf{sid})$ from $\mathcal{Z}$.
  - Monitoring $\bar{\mathcal{G}}_{\text{BB}}$, when a valid $(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ is posted on $\bar{\mathcal{G}}_{\text{BB}}$ from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, use $\mathsf{NIZK}_{\mathcal{R}_{11}}.\mathsf{Ext}^{\text{RO}}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corresponding secret key $\overline{\mathtt{sk}}_j$ and random coin α_j.
- In the ballot casting phase:
  - Upon receiving $(\textsc{CastNotify}, \mathsf{sid}, \mathsf{V}_i)$ from the external $\mathcal{F}_{\text{SV}}$ for an honest voter $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ creates $c_i \leftarrow \mathsf{TE.Enc}(\mathtt{pk}, 0)$. It then uses $\mathsf{NIZK}_{\mathcal{R}_{12}}.\mathsf{Sim}$ to simulate the corresponding proofs $\pi_i^{(2)}$. The simulator $\mathcal{S}$ then follows the protocol to post $(c_i, \pi_i^{(2)})$ to $\bar{\mathcal{G}}_{\text{BB}}$.
  - Monitoring $\bar{\mathcal{G}}_{\text{BB}}$, when a valid $(c_i, \pi_i^{(2)})$ is posted on $\bar{\mathcal{G}}_{\text{BB}}$ from a corrupted voter $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, uses the extracted $\{\overline{\mathtt{sk}}_j\}_{j \in [k]}$ to decrypt c_i to (V_i, s_i). The simulator $\mathcal{S}$ then acts as V_i to send $(\textsc{Cast}, \mathsf{sid}, s_i)$ to $\mathcal{F}_{\text{SV}}$.
- In the tally phase:
  - Upon receiving $(\textsc{Leak}, \mathsf{sid}, \tau)$ from the external $\mathcal{F}_{\text{SV}}$, the simulator $\mathcal{S}$ acts as the simulated $\mathcal{F}_{\text{MPC}}^{\mathcal{C}}$ to send τ to each of the trustees $\mathsf{T}_j \in \mathbb{T}$. For any honest trustee $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j to post τ on the $\bar{\mathcal{G}}_{\text{BB}}$.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_3$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}, \mathcal{F}_{\text{MPC}}^{\mathcal{C}}}_{\Pi_{\text{MPC-SV}}, \mathcal{A}, \mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that $\mathcal{H}_1$ runs $\mathsf{NIZK}_{\mathcal{R}_{11}}.\mathsf{Ext}^{\text{RO}}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corrupted trustee's secret key $\overline{\mathtt{sk}}_j$. $\mathcal{H}_1$ halt if the extraction fails.

▶ Claim 22. $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable.

Proof. According to Def. 4, the probability $\mathsf{NIZK}_{\mathcal{R}_{11}}.\mathsf{Ext}^{\text{RO}}$ extraction fails (a.k.a. knowledge error) is negligible, so the probability that any adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$ is $\mathsf{negl}(\lambda)$. ◀

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except the followings. During the vote phase, $\mathcal{H}_3$ uses $\mathsf{NIZK}_{\mathcal{R}_{12}}.\mathsf{Sim}$ to simulate $\pi_i^{(2)}$ for all the honest voter $\mathsf{V}_i \in \mathbb{V}$.

▶ Claim 23. $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable.

Proof. The advantage of the adversary is bounded by the ZK property of $\mathsf{NIZK}_{\mathcal{R}_{12}}$ as defined by Def. 3. ◀

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except the followings. During the vote phase, the simulator posts $c_i \leftarrow \mathsf{TE.Enc}(\mathtt{pk}, 0)$ for all the honest voter $\mathsf{V}_i \in \mathbb{V}$.

▶ Claim 24. $\mathcal{H}_3$ and $\mathcal{H}_2$ are indistinguishable.

Proof. The probability that any adversary $\mathcal{A}$ can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ is bounded by $\mathsf{AdvCPA}_{\mathcal{A}}(1^{\lambda})$ and ciphertext transformative indistinguishability. More specifically, we now show the if there exists an adversary $\mathcal{A}$ who can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$, then we can construction an adversary $\mathcal{B}$ that can break the IND-CPA game of the underlying TE by reduction. During the IND-CPA game, $\mathcal{B}$ receives a public key $\mathtt{pk}^*$ from the challenger. There must be at least one honest trustee in this case, and with our loss of generality, assume T_x is honest. During the preparation phase, $\mathcal{B}$ posts $\mathtt{pk}^*$ as T_x's public key together with simulated proof. During the ballot casting phase, for each honest voter V_i, $i \in [n]$, $\mathcal{B}$ sends $m_0 := 0$ and $m_1 := s_i$ to the IND-CPA challenger, and receives c^*. $\mathcal{B}$ then computes $c' \leftarrow \mathsf{TE.Trans}(c^*, \{\mathtt{sk}_i\}_{i\in[k]\setminus\{x\}})$. It posts c' as the honest voter's encrypted ballot. It is easy to see that, when c^* encrypts m_0, the adversary's view is indistinguishable from $\mathcal{H}_3$; when c^* encrypts m_1, the adversary's view is indistinguishable from $\mathcal{H}_2$. Hence, if $\mathcal{A}$ can distinguish $\mathcal{H}_3$ from $\mathcal{H}_2$ with non-negligible probability, then $\mathcal{B}$ can break the IND-CPA game with the same probability.

◀

The adversary's view of $\mathcal{H}_3$ is identical to the simulated view $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\mathrm{BB}}}_{\mathcal{F}_{\mathrm{SV}},\mathcal{S},\mathcal{Z}}$. Therefore, no PPT $\mathcal{Z}$ can distinguish the view of the ideal execution from the view of the real execution with more than negligible probability.

Case 2: $0 \le |\mathbb{V}_{\mathsf{corrupt}}| < n \ \wedge\ |\mathbb{T}_{\mathsf{corrupt}}| = k$.

Simulator. Similar as Case 1, the simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates honest voters $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, honest trustees $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$ and functionalities $\widehat{\mathcal{F}}_{\mathrm{CERT}}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- In the preparation phase:
  - Monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$, when a valid $(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, use $\mathsf{NIZK}_{\mathcal{R}_{11}}.\mathsf{Ext}^{\mathrm{RO}}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corresponding secret key $\overline{\mathtt{sk}}_j$.
- In the ballot casting phase:
  - Upon receiving $(\textsc{Leak}, \mathsf{sid}, \mathsf{V}_i, s_i)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest voter $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as V_i, following the protocol $\Pi_{\text{MPC-SV}}$ as if V_i receives $(\textsc{Cast}, \mathsf{sid}, s_i)$ from $\mathcal{Z}$.
  - Monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$, when a valid $(c_i, \pi_i^{(2)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted voter $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, uses the extracted $\left\{\overline{\mathtt{sk}}_j\right\}_{j\in[k]}$ to decrypt c_i to (V_i, s_i). The simulator $\mathcal{S}$ then acts as V_i to send $(\textsc{Cast}, \mathsf{sid}, s_i)$ to $\mathcal{F}_{\mathrm{SV}}$.
- In the tally phase:
  - Once the simulated $\mathcal{F}^{\mathcal{C}}_{\mathrm{MPC}}$ receives $(\textsc{Input}, \mathsf{sid}, \alpha_j, \{\mathtt{pk}_\ell\}_{\ell\in[k]}, \{c_i\}_{i\in[n]})$ from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j to send $(\textsc{Tally}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{SV}}$.

Indistinguishability. The indistinguishability in this case is straightforward, as $\mathcal{S}$ never simulate a single message to either any corrupted parties or the external $\bar{\mathcal{G}}_{\mathrm{BB}}$. The simulator $\mathcal{S}$ knows all the honest voters' ballot from the external $\mathcal{F}_{\mathrm{SV}}$, it simply acts as the honest voters according to the protocol $\Pi_{\text{MPC-SV}}$. Meanwhile, it also extracts the ballot of the malicious voters by using the extracted trustees' secret keys. Hence, the simulator $\mathcal{S}$ can submit the extracted ballot to the external $\mathcal{F}_{\mathrm{SV}}$ on the malicious voters' behave. Therefore, when NIZK extraction for trustees' secret keys are successful, the view of $\mathcal{Z}$ in the ideal execution has identical distribution to the view of $\mathcal{Z}$ in the real execution.

Case 3: $|\mathbb{V}_{\mathsf{corrupt}}| = n \ \wedge\ 0 \le |\mathbb{T}_{\mathsf{corrupt}}| \le k$.

Simulator. Trivial case. There is nothing needs to extract, as the trustees do not have input. The simulator $\mathcal{S}$ just run trustee according to protocol $\Pi_{\text{MPC-SV}}$.

Indistinguishability. The view of $\mathcal{Z}$ in the ideal execution has identical distribution to the view of of $\mathcal{Z}$ in the real execution.

◀

C.3 Threshold PKE

We would like to adopt a key-homomorphic threshold PKE scheme TE. It consists of a tuple of algorithms: $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CombinePK}, \mathsf{CombineSK}, \mathsf{ShareDec}, \mathsf{ShareCombine})$ as follows.

- $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$. The algorithm Setup takes input as the security parameter λ, and outputs public parameters param. All the other algorithms implicitly take param as input.
- $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{Keygen}(\mathsf{param})$. The algorithm Keygen takes input as the public parameter param, and outputs a public key $\mathtt{pk}$, a secret key $\mathtt{sk}$.
- $c \leftarrow \mathsf{Enc}(\mathtt{pk}, m)$. The algorithm Enc takes input as the public key $\mathtt{pk}$ and the message m, and outputs the ciphertext c.
- $c' \leftarrow \mathsf{ReRand}(\mathtt{pk}, c)$. The algorithm ReRand takes input as the public key $\mathtt{pk}$ and a ciphertext c, and outputs a re-randomized ciphertext c'.
- $m \leftarrow \mathsf{Dec}(\mathtt{sk}, c)$. The algorithm Dec takes input as the secret key $\mathtt{sk}$ and a ciphertext c, and outputs the decrypted plaintext m.
- $\mathtt{pk} := \mathsf{CombinePK}(\mathtt{pk}_1, \ldots, \mathtt{pk}_k)$. The algorithm $\mathsf{CombinePK}$ takes input as a set of public keys $(\mathtt{pk}_1, \ldots, \mathtt{pk}_k)$, and outputs a combined public key $\mathtt{pk}$.
- $\mathtt{sk} \leftarrow \mathsf{CombineSK}(\mathtt{sk}_1, \ldots, \mathtt{sk}_k)$. The algorithm $\mathsf{CombineSK}$ takes input as a set of secret key $(\mathtt{sk}_1, \ldots, \mathtt{sk}_k)$, and outputs combined secret key $\mathtt{sk}$.
- $\mu_i \leftarrow \mathsf{ShareDec}(\mathtt{sk}_i, c)$. The algorithm $\mathsf{ShareDec}$ takes input as the secret key $\mathtt{sk}_i$ and a ciphertext c, and outputs a decryption share μ_i.
- $m \leftarrow \mathsf{ShareCombine}(c, \mu_1, \ldots, \mu_k)$. The algorithm $\mathsf{ShareCombine}$ takes input as a ciphertext c and k decryption shares $(\mu_1, \ldots, \mu_k)$, and outputs a plaintext m.
- $c' \leftarrow \mathsf{Trans}(c, \{\mathtt{sk}_i\}_{i\in[k]\setminus\{j\}})$. The algorithm Trans takes input as a ciphertext $c \leftarrow \mathsf{TE.Enc}(\mathtt{pk}_j, m)$ and a set of secret keys $\{\mathtt{sk}_i\}_{i\in[k]\setminus\{j\}}$, and outputs a ciphertext c'.
- $\{\mu_j\}_{j\in[k]\setminus\mathcal{I}} \leftarrow \mathsf{SimShareDec}(c, m, \{\mu_i\}_{i\in\mathcal{I}})$. The algorithm $\mathsf{SimShareDec}$ takes as input a ciphertext c, a plaintext m, and a set of decryption shares $\{\mu_i\}_{i\in\mathcal{I}}$ and outputs a set of decryption shares $\{\mu_j\}_{j\in[k]\setminus\mathcal{I}}$. Here $\mathcal{I} \subsetneq [k]$.

▶ **Definition 25.** We say $\mathsf{TE} = \{\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CombinePK}, \mathsf{CombineSK}, \mathsf{ShareDec}, \mathsf{ShareCombine}\}$ is a *secure* key-homomorphic threshold public key encryption if the following properties hold:

Key combination correctness: If $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$ are all valid key pairs, $\mathtt{pk} := \mathsf{TE.CombinePK}(\{\mathtt{pk}_i\}_{i\in[k]})$ and $\mathtt{sk} := \mathsf{TE.CombineSK}(\{\mathtt{sk}_i\}_{i\in[k]})$, then $(\mathtt{pk}, \mathtt{sk})$ is also a valid key pair.
For all ciphertext $c \in \mathcal{C}_{\mathtt{pk}}$, where $\mathcal{C}_{\mathtt{pk}}$ is the ciphertext-space defined by $\mathtt{pk}$, we have

$$\mathsf{TE.Dec}(\mathtt{sk}, c) = \mathsf{TE.ShareCombine}(c, \mathsf{TE.ShareDec}(\mathtt{sk}_1, c), \ldots, \mathsf{TE.ShareDec}(\mathtt{sk}_k, c)) \ .$$

Ciphertext transformative indistinguishability: There exists a PPT algorithm Trans such that if $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$ are all valid key pairs, $\mathtt{pk} := \mathsf{TE.CombinePK}(\{\mathtt{pk}_i\}_{i\in[k]})$ and

$\mathtt{sk} := \mathsf{TE.CombineSK}(\{\mathtt{sk}_i\}_{i\in[k]})$, then for all message m, for any $j \in [k]$, the following holds.

$$\big(\mathsf{param}, \mathsf{TE.Trans}(c, \{\mathtt{sk}_i\}_{i\in[k]\setminus\{j\}})\big) \approx \big(\mathsf{param}, \mathsf{TE.Enc}(\mathtt{pk}, m)\big)$$

IND-CPA security: We say that a TE scheme achieves *indistinguishability under plaintext attacks (IND-CPA)* if for any PPT adversary $\mathcal{A}$ the following advantage AdvCPA is negligible.

$\underline{\textsc{Experiment}^{\mathsf{CPA}}(1^\lambda)}$

1. Run $\mathsf{param} \leftarrow \mathsf{TE.Setup}(1^\lambda)$.
2. Run $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{TE.Keygen}(\mathsf{param})$;
4. $\mathcal{A}(\mathtt{pk})$ outputs m_0, m_1 of equal length;
5. Pick $b \leftarrow \{0,1\}$; Run $c \leftarrow \mathsf{TE.Enc}(\mathtt{pk}, m_b)$;
6. $\mathcal{A}(c)$ outputs b^*; It returns 1 if $b = b^*$; else, returns 0.

We define the advantage of $\mathcal{A}$ as

$$\mathsf{AdvCPA}_{\mathcal{A}}(1^\lambda) = \left|\Pr[\textsc{Experiment}^{\mathsf{CPA}}(1^\lambda) = 1] - \frac{1}{2}\right| .$$

Share-simulation indistinguishability: We say TE scheme achieves *share-simulation indistinguishability* if there exists a PPT simulator $\mathsf{SimShareDec}$ such that for all valid key pairs $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$, all subsets $\mathcal{I} \subsetneq [k]$, all message m, the following two distributions are computationally indistinguishable:

$$\big(\mathsf{param}, c, \mathsf{SimShareDec}(c, m, \{\mu_i\}_{i\in\mathcal{I}})\big) \approx \big(\mathsf{param}, c, \{\mu_j\}_{j\in[k]\setminus\mathcal{I}}\big)$$

where $\mathsf{param} \leftarrow \mathsf{TE.Setup}(1^\lambda)$, $c \leftarrow \mathsf{TE.Enc}(\mathtt{pk}, m)$ and $\mu_j \leftarrow \mathsf{TE.ShareDec}(\mathtt{sk}_j, c)$ for $j \in [k] \setminus \mathcal{I}$.

D Supplemental material for Section 3

D.1 Proof for Theorem 1

Proof. To prove the theorem, we construct a simulator $\mathcal{S}$ such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}}_{\Pi_{\text{MIX-SV}}, \mathcal{A}, \mathcal{Z}}$ where the parties $\mathbb{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ and $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$ run protocol $\Pi_{\text{MIX-SV}}$ in the $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}\}$-hybrid world and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\text{BB}}}_{\mathcal{F}_{\text{SV}}, \mathcal{S}, \mathcal{Z}}$ where the parties interact with functionality $\mathcal{F}_{\text{SV}}$ in the $\bar{\mathcal{G}}_{\text{BB}}$-hybrid model and corrupted parties are controlled by the simulator $\mathcal{S}$. Let $\mathbb{V}_{\mathsf{corrupt}} \subseteq \mathbb{V}$ and $\mathbb{T}_{\mathsf{corrupt}} \subseteq \mathbb{T}$ be the set of corrupted voters and trustees, respectively. We consider following cases.

Case 1: $0 \leq |\mathbb{V}_{\mathsf{corrupt}}| < n \ \wedge \ 0 \leq |\mathbb{T}_{\mathsf{corrupt}}| < k$.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates honest voters $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, honest trustees $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$ and functionalities $\widehat{\mathcal{F}}_{\text{CERT}}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- In the preparation phase:

Functionality $\mathcal{F}_{\mathrm{SV}}$

The functionality $\mathcal{F}_{\mathrm{SV}}$ interacts with a set of voters $\mathbb{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, a set of trustees $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$, and the adversary $\mathcal{S}$. Let $\mathbb{V}_{\mathsf{honest}}$, $\mathbb{V}_{\mathsf{corrupt}}$ and $\mathbb{T}_{\mathsf{honest}}$, $\mathbb{T}_{\mathsf{corrupt}}$ denote the set of honest/corrupt voters and trustees, respectively.
Functionality $\mathcal{F}_{\mathrm{SV}}$ is parameterized by an algorithm $\mathsf{TallyProcess}$ (see Figure 2), a working table $\mathbb{W}$, and variables result, T_1, T_2, and B_i for all $i \in [n]$.
Initially, set $\mathit{result} := \emptyset$, $T_1 := \emptyset$, $T_2 := \emptyset$; for $i \in [n]$, set $B_i := \emptyset$.
Table $\mathbb{W}$ consists of n entries, and each entry consists of voter's real ID, voter's alternative ID, and the statement that the voter submitted; for all $i \in [n]$, the ith entry $\mathbb{W}[i] := (\mathsf{V}_i, w_i, \mathit{statement}_i)$, where $w_i \leftarrow \{0,1\}^\lambda$, $\mathit{statement}_i := \emptyset$.

Preparation:

1. Upon receiving input $(\textsc{InitialTrustee}, \mathsf{sid})$ from the trustee $\mathsf{T}_j \in \mathbb{T}$, set $T_1 := T_1 \cup \{\mathsf{T}_j\}$, and send a notification message $(\textsc{InitialTrusteeNotify}, \mathsf{sid}, \mathsf{T}_j)$ to the adversary $\mathcal{S}$.

Ballot Casting:

1. Upon receiving input $(\textsc{Cast}, \mathsf{sid}, (s_i, w_i^*))$ from the voter $\mathsf{V}_i \in \mathbb{V}$, if $|T_1| < k$, ignore the input.
   Otherwise,
   - if V_i is honest (now $w_i^* := \perp$), then update $\mathbb{W}[i] := (\mathsf{V}_i, w_i, s_i)$; send a message $(\textsc{CastNotify}, \mathsf{sid}, \mathsf{V}_i)$ to the adversary $\mathcal{S}$.
   - if V_i is corrupt, then update $\mathbb{W}[i] := (\mathsf{V}_i, w_i^*, s_i)$.

   If $|\mathbb{T}_{\mathsf{corrupt}}| = k$, then additionally send a message $(\textsc{Leak}, \mathsf{sid}, \mathbb{W}[i])$ to the adversary $\mathcal{S}$.

Tally:

1. Upon receiving input $(\textsc{Tally}, \mathsf{sid})$ from the trustee $\mathsf{T}_j \in \mathbb{T}$, set $T_2 := T_2 \cup \{\mathsf{T}_j\}$ and do the following:
   - set $\mathbb{U} := \mathbb{W}$; then eliminate all V_i's in $\mathbb{U}$; finally sort the entries in $\mathbb{U}$ lexicographically.

   Send a notification message $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_j)$ to $\mathcal{S}$.
   If $|T_2 \cap \mathbb{T}_{\mathsf{honest}}| + |\mathbb{T}_{\mathsf{corrupt}}| = k$, send a leakage message $(\textsc{Leak}, \mathsf{sid}, \mathbb{U})$ to $\mathcal{S}$.
   If $|T_2| = k$, compute $\mathit{result} \leftarrow \mathsf{TallyProcess}(\mathbb{U})$.
2. Upon receiving input $(\textsc{ReadResult}, \mathsf{sid})$ from a voter $\mathsf{V}_i \in \mathbb{V}$, if $\mathit{result} = \emptyset$, ignore the input.
   Else, return $(\textsc{ResultReturn}, \mathsf{sid}, \mathit{result})$ to V_i.

Figure 17 The voting functionality $\mathcal{F}_{\mathrm{SV}}$.

- Upon receiving $(\textsc{InitialTrusteeNotify}, \mathsf{sid}, \mathsf{T}_j)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest trustee $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j, following the protocol $\Pi_{\text{MIX-SV}}$ as if T_j receives $(\textsc{InitialTrustee}, \mathsf{sid})$ from $\mathcal{Z}$.
- Monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$, when a valid $(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, use $\mathsf{NIZK}_{\mathcal{R}_4}.\mathsf{Ext}^{\mathrm{RO}}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corresponding secret key $\overline{\mathtt{sk}}_j$.

In the ballot casting phase:

- Upon receiving $(\textsc{CastNotify}, \mathsf{sid}, \mathsf{V}_i)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest voter $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as V_i, following the protocol $\Pi_{\text{MIX-SV}}$ round 1 description as if V_i receives $(\textsc{Cast}, \mathsf{sid}, (\cdot, \perp))$ from $\mathcal{Z}$. In round 2, the simulator $\mathcal{S}$ creates $U_{i,\ell} \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, 0)$, $\ell \in [\lambda_1]$ and $S_i \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, 0)$. It then simulates the corresponding proofs $\pi_{i,\ell}^{(3)}$ and $\pi_i^{(4)}$. The simulator $\mathcal{S}$ then follows the protocol to post $(U_{i,\ell}, \pi_{i,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_i, \pi_i^{(4)})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$.
- The simulator $\mathcal{S}$ monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$; once a $(W_i, \pi_i^{(2)})$ is posted from a corrupted voter $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ uses the extracted $\{\overline{\mathtt{sk}}_j\}_{j \in [k]}$ to decrypt W_i to the temporal ID w_i. Record (V_i, w_i). When a valid $(U_{i,\ell}, \pi_{i,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_i, \pi_i^{(4)})$ is posted on

  $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted voter $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, uses the extracted $\{\overline{\mathtt{sk}}_j\}_{j\in[k]}$ to decrypt $U_{i,\ell}$ to $w_{i,\ell}$ and S_i to s_i. Replace the ID references in s_i to their actuarial voter ID's, and denoted the modified statement as s'_i. Record (V_i, s'_i).
  - Upon receiving any $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_j)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest trustee $\mathsf{T}_j \in \mathbb{T}\backslash\mathbb{T}_{\mathsf{corrupt}}$ or any corrupted trustee has moved to the tally phase, the simulator $\mathcal{S}$ acts as each of the corrupted voters $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$ to send $(\textsc{Cast}, \mathsf{sid}, (s'_i, w_i))$ to $\mathcal{F}_{\mathrm{SV}}$ if both (V_i, w_i) and (V_i, s'_i) is recorded; otheriwse, it acts as V_i to send $(\textsc{Cast}, \mathsf{sid}, (s'_i, \perp))$ to $\mathcal{F}_{\mathrm{SV}}$ if only (V_i, s'_i) is recorded.
- In the tally phase:
  - Upon receiving $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_j)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest trustee $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$, if $\{\overline{m}^{(j)}_{i,\ell}\}_{i\in[n'],\ell\in[\lambda_1+2]}$ is not defined, the simulator $\mathcal{S}$ acts as T_j, following the protocol $\Pi_{\text{MIX-SV}}$ as if T_j receives $(\textsc{Tally}, \mathsf{sid})$ from $\mathcal{Z}$. $\mathcal{S}$ then adds j to $\mathcal{J}$, where $\mathcal{J}$ is initially empty. If $\{\overline{m}^{(j)}_{i,\ell}\}_{i\in[n'],\ell\in[\lambda_1+2]}$ is defined, $\mathcal{S}$ uses $\mathsf{NIZK}_{\mathcal{R}_8}.\mathsf{Sim}$ to simulate the corresponding proof $\pi^{(6)}_{i,j,\ell}$. It then follows the protocol to post $(\overline{m}^{(j)}_{i,\ell}, \pi^{(6)}_{i,j,\ell})_{i\in[n'],\ell\in[\lambda_1+2]}$ on the $\bar{\mathcal{G}}_{\mathrm{BB}}$.
  - The simulator $\mathcal{S}$ monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$; once $(\overline{m}^{(j)}_{i,\ell}, \pi^{(6)}_{i,j,\ell})_{i\in[n'],\ell\in[\lambda_1+2]}$ is posted from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j to send $(\textsc{Tally}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{SV}}$.
  - Upon receiving $(\textsc{Leak}, \mathsf{sid}, (\tilde{B}_1, \ldots, \tilde{B}_n))$ from the external $\mathcal{F}_{\mathrm{SV}}$, the simulator $\mathcal{S}$ uses the extracted secret key $\overline{\mathtt{sk}}_j$ to compute $\overline{m}^{(j)}_{i,\ell} \leftarrow \mathsf{TRE.ShareDec}(\overline{\mathtt{sk}}_j, e^{(k)}_{i,\ell})$ for all the corrupted trustees $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$. The simulator $\mathcal{S}$ then uses $\mathsf{TRE.SimShareDec}$. to compute the message shares of the rest honest T_j's message shares $\overline{m}^{(j)}_{i,\ell}$ according to $(\tilde{B}_1, \ldots, \tilde{B}_n)$.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_4$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\mathrm{CERT}}}_{\Pi_{\text{MIX-SV}}, \mathcal{A}, \mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that $\mathcal{H}_1$ runs $\mathsf{NIZK}_{\mathcal{R}_4}.\mathsf{Ext}^{\mathrm{RO}}(\overline{\mathtt{pk}}_j, \pi^{(1)}_j)$ to extract the corrupted trustee's secret key $\overline{\mathtt{sk}}_j$. $\mathcal{H}_1$ halt if the extraction fails.

▶ Claim 26. $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable.

Proof. According to Def. 4, the probability $\mathsf{Ext}^{\mathrm{RO}}$ extraction fails (a.k.a. knowledge error) is negligible, so the probability that any adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$ is $\mathsf{negl}(\lambda)$. ◀

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except the following: During the tally phase, uses the extracted $\mathtt{sk}_j$ from Hybrid $\mathcal{H}_1$ to decrypt each ciphertext, and the last honest trustee's message shares of each ciphertext are calculated by $\mathsf{TRE.SimShareDec}$ instead of using $\mathsf{TRE.ShareDec}$.

▶ Claim 27. $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable.

Proof. By the share-simulation indistinguishability of the underlying TRE scheme, the distribution of the simulated decryption share(s) are computationally indistinguishable to the real ones. Moreover, by soundness of

$$\pi^{(6)}_{i,j,\ell} \leftarrow \mathsf{NIZK}_{\mathcal{R}_8}\left\{\begin{array}{c} (e^{(k)}_{i,\ell}, \overline{m}^{(j)}_{i,\ell}, \overline{\mathtt{pk}}_j), (\overline{\mathtt{sk}}_j, \alpha_j): \\ (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{TRE.Keygen}(\mathsf{param}; \alpha_j) \\ \wedge\, \overline{m}^{(j)}_{i,\ell} = \mathsf{TRE.ShareDec}(\overline{\mathtt{sk}}_j, e^{(k)}_{i,\ell}) \end{array}\right\}$$

the corrupted trustees have negligible probability to post an invalid decryption share that is different from $\overline{m}_{i,\ell}^{(j)} \leftarrow \mathsf{TRE.ShareDec}(\overline{\mathtt{sk}}_j, e_{i,\ell}^{(k)})$. Therefore, the adversary's advantage of distinguishing $\mathcal{H}_2$ from $\mathcal{H}_1$ is $\mathsf{negl}(\lambda)$. ◀

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except the followings. During the vote phase, $\mathcal{H}_3$ uses $\mathsf{NIZK}_{\mathcal{R}_6}.\mathsf{Sim}$ to simulate $\pi_{i,\ell}^{(3)}$, $\ell \in [\lambda_1]$ and uses $\mathsf{NIZK}_{\mathcal{R}_7}.\mathsf{Sim}$ to simulate $\pi_i^{(4)}$ for all the honest voter $\mathsf{V}_i \in \mathbb{V}$.

▶ Claim 28. $\mathcal{H}_3$ and $\mathcal{H}_2$ are indistinguishable.

Proof. The advantage of the adversary is bounded by the ZK property of NIZK as defined by Def. 3. ◀

Hybrid $\mathcal{H}_4$: $\mathcal{H}_4$ is the same as $\mathcal{H}_3$ except the followings. During the vote phase, the simulator posts $U_{i,\ell} \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, 0)$, $\ell \in [\lambda_1]$ and $S_i \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, 0)$ for all the honest voter $\mathsf{V}_i \in \mathbb{V}$.

▶ Claim 29. $\mathcal{H}_4$ and $\mathcal{H}_3$ are indistinguishable.

Proof. The probability that any adversary $\mathcal{A}$ can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ is bounded by $\mathsf{AdvCPA}_{\mathcal{A}}(1^\lambda)$, $\mathsf{AdvUnlink}_{\mathcal{A}}(1^\lambda)$ and ciphertext transformative indistinguishability. More specifically, we now show the if there exists an adversary $\mathcal{A}$ who can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$, then we can construction an adversary $\mathcal{B}$ that can break the IND-CPA game of the underlying TRE by reduction. During the IND-CPA game, $\mathcal{B}$ receives a public key $\mathtt{pk}^*$ from the challenger. There must be at least one honest trustee in this case, and with our loss of generality, assume T_x is honest. During the preparation phase, $\mathcal{B}$ posts $\mathtt{pk}^*$ as T_x's public key together with simulated proof. During the ballot casting phase, for each honest voter V_i, $i \in [n]$, $\mathcal{B}$ sends $m_0 := (0, 0, \ldots, 0)$ and $m_1 := (w_{i,1}, \ldots, w_{i,\lambda_1}, s_i)$ to the IND-CPA challenger, and receives $\{c_\ell^*\}_{\ell \in [\lambda_1+1]}$. $\mathcal{B}$ then computes $c'_\ell \leftarrow \mathsf{TRE.Trans}(c_\ell^*, \{\mathtt{sk}_i\}_{i \in [k] \setminus \{x\}})$. It posts c' as the honest voter's encrypted ballot. It is easy to see that, due to $\mathsf{AdvUnlink}_{\mathcal{A}}(1^\lambda)$, when $\{c_\ell^*\}_{\ell \in [\lambda_1+1]}$ encrypts m_0, the adversary's view is indistinguishable from $\mathcal{H}_4$; when $\{c_\ell^*\}_{\ell \in [\lambda_1+1]}$ encrypts m_1, the adversary's view is indistinguishable from $\mathcal{H}_3$. Hence, if $\mathcal{A}$ can distinguish $\mathcal{H}_4$ from $\mathcal{H}_3$ with non-negligible probability, then $\mathcal{B}$ can break the IND-CPA game with the same probability.

◀

The adversary's view of $\mathcal{H}_4$ is identical to the simulated view $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\mathrm{BB}}}_{\mathcal{F}_{\mathrm{SV}},\mathcal{S},\mathcal{Z}}$. Therefore, no PPT $\mathcal{Z}$ can distinguish the view of the ideal execution from the view of the real execution with more than negligible probability.

Case 2: $0 \leq |\mathbb{V}_{\mathsf{corrupt}}| < n \ \wedge \ |\mathbb{T}_{\mathsf{corrupt}}| = k$.

Simulator. Similar as Case 1, the simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates honest voters $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, honest trustees $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$ and functionalities $\widehat{\mathcal{F}}_{\mathrm{CERT}}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- In the preparation phase:
  - Monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$, when a valid $(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, use $\mathsf{NIZK}_{\mathcal{R}_4}.\mathsf{Ext}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corresponding secret key $\overline{\mathtt{sk}}_j$.
- In the ballot casting phase:

- Upon receiving $(\textsc{Leak}, \mathsf{sid}, \mathsf{V}_i, B_i)$ from the external $\mathcal{F}_{\mathrm{SV}}$ for an honest voter $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as V_i, following the protocol $\Pi_{\textsc{mix-sv}}$ as if V_i receives $(\textsc{Cast}, \mathsf{sid}, B_i)$ from $\mathcal{Z}$.
- The simulator $\mathcal{S}$ monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$; once a $(W_i, \pi_i^{(2)})$ is posted from a corrupted voter $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ uses the extracted $\{\overline{\mathtt{sk}}_j\}_{j\in[k]}$ to decrypt W_i to the temporal ID w_i. Record (V_i, w_i). When a valid $(U_{i,\ell}, \pi_{i,\ell}^{(3)})_{\ell=1}^{\lambda_1}, S_i, \pi_i^{(4)})$ is posted on $\bar{\mathcal{G}}_{\mathrm{BB}}$ from a corrupted voter $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, uses the extracted $\{\overline{\mathtt{sk}}_j\}_{j\in[k]}$ to decrypt $U_{i,\ell}$ to $w_{i,\ell}$ and S_i to s_i. Replace the ID references in s_i to their actuarial voter ID's, and denoted the modified statement as s_i'. Record (V_i, s_i').
- When any corrupted trustee has moved to the tally phase, the simulator $\mathcal{S}$ acts as each of the corrupted voters $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$ to send $(\textsc{Cast}, \mathsf{sid}, (s_i', w_i))$ to $\mathcal{F}_{\mathrm{SV}}$ if both (V_i, w_i) and (V_i, s_i') is recorded; otheriwse, it acts as V_i to send $(\textsc{Cast}, \mathsf{sid}, (s_i', \perp))$ to $\mathcal{F}_{\mathrm{SV}}$ if only (V_i, s_i') is recorded.

- In the tally phase:
  - The simulator $\mathcal{S}$ monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$; once $(\overline{m}_{i,\ell}^{(j)}, \pi_{i,j,\ell}^{(6)})_{i\in[n'],\ell\in[\lambda_1+2]}$ is posted from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j to send $(\textsc{Tally}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{SV}}$.

Indistinguishability. The indistinguishability in this case is straightforward, as $\mathcal{S}$ never simulate a single message to either any corrupted parties or the external $\bar{\mathcal{G}}_{\mathrm{BB}}$. The simulator $\mathcal{S}$ knows all the honest voters' ballot from the external $\mathcal{F}_{\mathrm{SV}}$, it simply acts as the honest voters according to the protocol $\Pi_{\textsc{mix-sv}}$. Meanwhile, it also extracts the ballot of the malicious voters by using the extracted trustees' secret keys. Hence, the simulator $\mathcal{S}$ can submit the extracted ballot to the external $\mathcal{F}_{\mathrm{SV}}$ on the malicious voters' behave. Therefore, when NIZK extraction for trustees' secret keys are successful, the view of $\mathcal{Z}$ in the ideal execution has identical distribution to the view of $\mathcal{Z}$ in the real execution.

Case 3: $|\mathbb{V}_{\mathsf{corrupt}}| = n \ \wedge \ 0 \le |\mathbb{T}_{\mathsf{corrupt}}| \le k$.

Simulator. Trivial case. There is nothing needs to extract, as the trustees do not have input. The simulator $\mathcal{S}$ just run trustee according to protocol $\Pi_{\textsc{mix-sv}}$.

Indistinguishability. The view of $\mathcal{Z}$ in the ideal execution has identical distribution to the view of $\mathcal{Z}$ in the real execution.

◀

D.2 Threshold re-randomizable encryption

A threshold re-randomizable encryption scheme TRE consists of a tuple of algorithms: (Setup, Keygen, Enc, Dec, CombinePK, CombineSK, ShareDec, ShareCombine, ReRand) as follows.

- $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$. The algorithm Setup takes input as the security parameter λ, and outputs public parameters param. All the other algorithms implicitly take param as input.
- $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{Keygen}(\mathsf{param})$. The algorithm Keygen takes input as the public parameter param, and outputs a public key pk, a secret key sk.
- $c \leftarrow \mathsf{Enc}(\mathtt{pk}, m)$. The algorithm Enc takes input as the public key pk and the message m, and outputs the ciphertext c.
- $c' \leftarrow \mathsf{ReRand}(\mathtt{pk}, c)$. The algorithm ReRand takes input as the public key pk and a ciphertext c, and outputs a re-randomized ciphertext c'.

- $m \leftarrow \mathsf{Dec}(\mathtt{sk}, c)$. The algorithm Dec takes input as the secret key $\mathtt{sk}$ and a ciphertext c, and outputs the decrypted plaintext m.
- $\mathtt{pk} := \mathsf{CombinePK}(\mathtt{pk}_1, \ldots, \mathtt{pk}_k)$. The algorithm $\mathsf{CombinePK}$ takes input as a set of public keys $(\mathtt{pk}_1, \ldots, \mathtt{pk}_k)$, and outputs a combined public key $\mathtt{pk}$.
- $\mathtt{sk} \leftarrow \mathsf{CombineSK}(\mathtt{sk}_1, \ldots, \mathtt{sk}_k)$. The algorithm $\mathsf{CombineSK}$ takes input as a set of secret key $(\mathtt{sk}_1, \ldots, \mathtt{sk}_k)$, and outputs combined secret key $\mathtt{sk}$.
- $\mu_i \leftarrow \mathsf{ShareDec}(\mathtt{sk}_i, c)$. The algorithm $\mathsf{ShareDec}$ takes input as the secret key $\mathtt{sk}_i$ and a ciphertext c, and outputs a decryption share μ_i.
- $m \leftarrow \mathsf{ShareCombine}(c, \mu_1, \ldots, \mu_k)$. The algorithm $\mathsf{ShareCombine}$ takes input as a ciphertext c and k decryption shares $(\mu_1, \ldots, \mu_k)$, and outputs a plaintext m.
- $c' \leftarrow \mathsf{Trans}(c, \{\mathtt{sk}_i\}_{i\in[k]\setminus\{j\}})$. The algorithm Trans takes input as a ciphertext $c \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}_j, m)$ and a set of secret keys $\{\mathtt{sk}_i\}_{i\in[k]\setminus\{j\}}$, and outputs a ciphertext c'.
- $\{\mu_j\}_{j\in[k]\setminus\mathcal{I}} \leftarrow \mathsf{SimShareDec}(c, m, \{\mu_i\}_{i\in\mathcal{I}})$. The algorithm $\mathsf{SimShareDec}$ takes as input a ciphertext c, a plaintext m, and a set of decryption shares $\{\mu_i\}_{i\in\mathcal{I}}$ and outputs a set of decryption shares $\{\mu_j\}_{j\in[k]\setminus\mathcal{I}}$. Here $\mathcal{I} \subsetneq [k]$.

▶ **Definition 30.** We say $\mathsf{TRE} = \{\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CombinePK}, \mathsf{CombineSK}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{ReRand}\}$ is a *secure* threshold re-randomizable public key encryption if the following properties hold:

Key combination correctness: If $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$ are all valid key pairs, $\mathtt{pk} := \mathsf{TRE.CombinePK}(\{\mathtt{pk}_i\}_{i\in[k]})$ and $\mathtt{sk} := \mathsf{TRE.CombineSK}(\{\mathtt{sk}_i\}_{i\in[k]})$, then $(\mathtt{pk}, \mathtt{sk})$ is also a valid key pair.
For all ciphertext $c \in \mathcal{C}_{\mathtt{pk}}$, where $\mathcal{C}_{\mathtt{pk}}$ is the ciphertext-space defined by $\mathtt{pk}$, we have

$$\mathsf{TRE.Dec}(\mathtt{sk}, c) = \mathsf{TRE.ShareCombine}(c, \mathsf{TRE.ShareDec}(\mathtt{sk}_1, c), \ldots, \mathsf{TRE.ShareDec}(\mathtt{sk}_k, c)) \ .$$

Ciphertext transformative indistinguishability: There exists a PPT algorithm Trans such that if $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$ are all valid key pairs, $\mathtt{pk} := \mathsf{TRE.CombinePK}(\{\mathtt{pk}_i\}_{i\in[k]})$ and $\mathtt{sk} := \mathsf{TRE.CombineSK}(\{\mathtt{sk}_i\}_{i\in[k]})$, then for all message m, for any $j \in [k]$, the following holds.

$$\big(\mathsf{param}, \mathsf{TRE.Trans}(c, \{\mathtt{sk}_i\}_{i\in[k]\setminus\{j\}})\big) \approx \big(\mathsf{param}, \mathsf{TRE.Enc}(\mathtt{pk}, m)\big)$$

IND-CPA security: We say that a TRE scheme achieves *indistinguishability under plaintext attacks (IND-CPA)* if for any PPT adversary $\mathcal{A}$ the following advantage AdvCPA is negligible.

$\underline{\text{EXPERIMENT}^{\mathsf{CPA}}(1^\lambda)}$

1. Run $\mathsf{param} \leftarrow \mathsf{TRE.Setup}(1^\lambda)$.
2. Run $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{TRE.Keygen}(\mathsf{param})$;
4. $\mathcal{A}(\mathtt{pk})$ outputs m_0, m_1 of equal length;
5. Pick $b \leftarrow \{0, 1\}$; Run $c \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, m_b)$;
6. $\mathcal{A}(c)$ outputs b^*; It returns 1 if $b = b^*$; else, returns 0.

We define the advantage of $\mathcal{A}$ as

$$\mathsf{AdvCPA}_{\mathcal{A}}(1^\lambda) = \left|\Pr[\text{EXPERIMENT}^{\mathsf{CPA}}(1^\lambda) = 1] - \frac{1}{2}\right| \ .$$

Unlinkability: We say a TRE scheme is *unlinkable* if for any PPT adversary $\mathcal{A}$ the following advantage $\mathsf{AdvUnlink}$ is negligible.

$\underline{\text{EXPERIMENT}^{\mathsf{Unlink}}(1^\lambda)}$

1. $\mathcal{A}$ outputs a set $\mathcal{I} \subset \{1, \ldots, k\}$ of up to $k-1$ corrupted indices.
2. For $i = [n]$, run $(\overline{\mathtt{pk}}_i, \overline{\mathtt{sk}}_i) \leftarrow \mathsf{TRE.Keygen}(1^\lambda; \omega_i)$;
3. $\mathcal{A}(\{\mathtt{pk}_j\}_{j\in[k]\setminus\mathcal{I}})$ outputs c_0, c_1;
4. $b \leftarrow \{0,1\}$; $c' \leftarrow \mathsf{TRE.ReRand}(\mathtt{pk}, c_b; \omega)$;
5. $\mathcal{A}(c')$ outputs b^*; It returns 1 if $b = b^*$; else, returns 0.

We define the advantage of $\mathcal{A}$ as

$$\mathsf{AdvUnlink}_{\mathcal{A}}(1^\lambda) = \left| \Pr[\text{EXPERIMENT}^{\mathsf{Unlink}}(1^\lambda) = 1] - \frac{1}{2} \right| .$$

Share-simulation indistinguishability: We say TRE scheme achieves *share-simulation indistinguishability* if there exists a PPT simulator SimShareDec such that for all valid key pairs $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}_{i\in[k]}$, all subsets $\mathcal{I} \subsetneq [k]$, all message m, the following two distributions are computationally indistinguishable:

$$\big(\mathsf{param}, c, \mathsf{SimShareDec}(c, m, \{\mu_i\}_{i\in\mathcal{I}})\big) \approx \big(\mathsf{param}, c, \{\mu_j\}_{j\in[k]\setminus\mathcal{I}}\big)$$

where $\mathsf{param} \leftarrow \mathsf{TRE.Setup}(1^\lambda)$, $c \leftarrow \mathsf{TRE.Enc}(\mathtt{pk}, m)$ and $\mu_j \leftarrow \mathsf{TRE.ShareDec}(\mathtt{sk}_j, c)$ for $j \in [k] \setminus \mathcal{I}$.

D.3 Instantiation of TRE

We adopt threshold ElGamal encryption as a candidate for the threshold re-randomizable encryption (TRE) scheme. For any given security parameter λ, we pick a cyclic group $\langle g \rangle = \mathbb{G}$ with prime order q where the DDH assumption holds. The group information is denoted as param and is an implicit input of every algorithm.

- $\mathsf{TRE.Keygen}(\mathsf{param})$: The algorithm randomly picks $\overline{\mathtt{sk}}_i \leftarrow \mathbb{Z}_q$ and outputs $(\overline{\mathtt{pk}}_i := g^{\overline{\mathtt{sk}}_i}, \overline{\mathtt{sk}}_i)$.
- $\mathsf{TRE.CombinePK}(\{\overline{\mathtt{pk}}_i\}_{i=1}^k)$: The algorithm sets $h := \prod_{i=1}^k \overline{\mathtt{pk}}_i$ and outputs $\mathtt{pk} := (h, \overline{\mathtt{pk}}_1, \ldots, \overline{\mathtt{pk}}_k)$.
- $\mathsf{TRE.CombineSK}(\overline{\mathtt{sk}}_1, \ldots, \overline{\mathtt{sk}}_k)$. The algorithm CombineSK takes input as a set of secret key $(\overline{\mathtt{sk}}_1, \ldots, \overline{\mathtt{sk}}_k)$, and outputs combined secret key $\mathtt{sk} := \sum_{i=1}^k \overline{\mathtt{sk}}_i$.
- $\mathsf{TRE.Enc}(\mathtt{pk}, m)$: The algorithm randomly picks $r \leftarrow \mathbb{Z}_q$ and outputs $e := (g^r, m \cdot h^r)$.
- $\mathsf{TRE.ReRand}(\mathtt{pk}, e)$: The algorithm first parses e into (e_1, e_2), then randomly picks $s \leftarrow \mathbb{Z}_q$ and outputs $e' := (g^s \cdot e_1, h^s \cdot e_2)$.
- $\mathsf{TRE.Dec}(\mathtt{sk}, e)$. The algorithm Dec first parses e into (e_1, e_2), and outputs the decrypted plaintext $m := e_2 / e_1^{\mathtt{sk}}$.
- $\mathsf{TRE.ShareDec}(\mathtt{pk}, \overline{\mathtt{sk}}_i, e)$: The algorithm first parses ciphertext e into (e_1, e_2); then it outputs $\overline{m}_i := e_1^{\overline{\mathtt{sk}}_i}$.
- $\mathsf{TRE.ShareCombine}(e, \{\overline{m}_i\}_{i=1}^k)$: The algorithm first parses ciphertext e into (e_1, e_2); then it outputs $m := e_2 / \prod_{i=1}^k \overline{m}_i$.
- $\mathsf{Trans}(e, \{\mathtt{sk}_i\}_{i\in[k]\setminus\{j\}})$. The algorithm first parses e into (e_1, e_2); then it outputs $(e_1, e_2 \cdot \prod_{i\in[k]\setminus\{j\}} e_1^{\mathtt{sk}_i})$.
- $\mathsf{SimShareDec}(e, m, \{\mu_i\}_{i\in\mathcal{I}})$. The algorithm first parses e into (e_1, e_2) and then generates random decryption shares $\{\mu_j\}_{j\in[k]\setminus\mathcal{I}}$ except for the last one, denoted as μ_x. It then set $\mu_x = \frac{e_2}{m \cdot \prod_{j\in[k]\setminus\{x\}} \mu_j}$ and outputs $\{\mu_j\}_{j\in[k]\setminus\mathcal{I}}$.

First of all, the correctness of the above scheme follows by inspection. Now let's examine the security properties. It is easy to see that $\mathsf{AdvCPA}_{\mathcal{A}}(1^\lambda) = \mathsf{negl}(\lambda)$ is guaranteed by the IND-CPA security of the underlying ElGamal encryption which is under the DDH assumption. Besides, $\mathsf{AdvUnlink}_{\mathcal{A}}(1^\lambda) = 0$, as each re-randomized ciphertext has the same distribution as a freshly encrypted ciphertext. In terms of the ciphertext transformative indistinguishability, it is perfectly indistinguishable as the resulting ciphertext has the same distribution as a freshly encrypted one. Finally, share-simulation indistinguishability is also straightforward and it is implied by IND-CPA security.

D.4 Instantiations of NIZKs

Several NIZK proofs are used in our construction. Hereby, we provide RO-based instantiation for these primitives.

NIZK for distributed key generation. In the preparation phase, we used a NIZK proof of knowledge for knowledge of the secret key and correctness of the distributed key generation, i.e.,

$$\mathsf{NIZK}_{\mathcal{R}_4}\big\{(\overline{\mathtt{pk}}), (\omega, \overline{\mathtt{sk}}) : (\overline{\mathtt{pk}}, \overline{\mathtt{sk}}) = \mathsf{TRE.Keygen}(\mathsf{param}; \omega)\big\}$$

In terms of ElGamal encryption, this NIZK can be realized by strong Fiat-Shamir heuristic of the Schnorr's proof [46]. Schnorr's proof is Sigma proof of knowledge of discrete logarithm; however, its RO-NIZK version has a small caveat, i.e., the knowledge extraction is based on RO rewinding. Alternatively, to enable extractability, we propose to a NIZK in Fig. 18, where $H_1 : \{0,1\}^* \mapsto \mathbb{G}$ is a hash function. $\mathsf{NIZK}_{\mathcal{R}_9}$ allows the prover to show an ElGamal ciphertext is encryption of 0/1 using a Sigma disjunction of Chaum-Pederden Sigma protocol. $\mathsf{NIZK}_{\mathcal{R}_{10}}$ is strong Fiat-Shamir heuristic of Chaum-Pederden Sigma protocol for DDH tuples.

NIZK for Discrete Logarithm

Statement: $h = g^s$
Witness: $s_1, \ldots, s_\kappa \in \{0,1\}$ s.t. $s = \sum_{i=1}^{\kappa} 2^{i-1} s_i$
Prove:

- Set $u := H_1(h)$ and pick $r_1, \ldots, r_\kappa \leftarrow \mathbb{Z}_q$.
- For $i \in [\kappa]$, compute $e_{i,1} := g^{r_1}$, $e_{i,2} := g^{s_i} u^{r_1}$, and prove

$$\pi_i \leftarrow \mathsf{NIZK}_{\mathcal{R}_9}\Big\{\ (g, u, e_{i,1}, e_{i,2}), (s_i, r_i) : (e_{i,1} = g^{r_i} \wedge e_{i,2} = u^{r_i})\ \vee (e_{i,1} = g^{r_i} \wedge e_{i,2}/g = u^{r_i})\ \Big\}\ .$$

- Compute $e_1 := \prod_{i=1}^{\kappa} (e_{i,1})^{2^{i-1}}$, $e_2 := \prod_{i=1}^{\kappa} (e_{i,2})^{2^{i-1}}$ and $r := \sum_{i=1}^{\kappa} 2^{i-1} r_i$. Prove

$$\phi \leftarrow \mathsf{NIZK}_{\mathcal{R}_{10}}\big\{(g, u, e_1, e_2), (r) : (e_1 = g^r \wedge e_2/h = u^r)\big\}\ .$$

- Output $\pi := ((e_{i,1}, e_{i,2})_{i\in[\kappa]}, \pi_1, \ldots, \pi_\kappa, \phi)$.

Verify:

- Set $u := H_1(h)$, $e_1 := \prod_{i=1}^{\kappa} (e_{i,1})^{2^{i-1}}$, and $e_2 := \prod_{i=1}^{\kappa} (e_{i,2})^{2^{i-1}}$.
- For $i \in [\kappa]$, check $\mathsf{NIZK}_{\mathcal{R}_9}.\mathsf{Verify}\big\{(g, u, e_{i,1}, e_{i,2}), \pi_i\big\}$.
- Check $\mathsf{NIZK}_{\mathcal{R}_{10}}.\mathsf{Verify}\big\{(g, u, e_1, e_2), \phi\big\}$.

Figure 18 NIZK for Discrete Logarithm.

▶ **Theorem 31.** *The* NIZK *described in Fig. 18 is an NIZK proof of knowledge of $s \in \mathbb{Z}_q$ for $h = g^s$ with extractability.*

Proof. The completeness and soundness follow directly by the completeness of the underlying $\mathsf{NIZK}_{\mathcal{R}_9}$ and $\mathsf{NIZK}_{\mathcal{R}_{10}}$. For ZK, the simulator generates $(e_{i,1}, e_{i,2})$ as encryption of 0 and computes $\mathsf{NIZK}_{\mathcal{R}_9}$ honestly. It then simulates ϕ using $\mathsf{NIZK}_{\mathcal{R}_{10}}.\mathsf{Sim}$. In terms of extractability, the knowledge extractor simulates the RO for H_1, and it outputs $u = g^x$ for a randomly chosen $x \in \mathbb{Z}_q$. Now the extractor can decrypt $(e_{i,1}, e_{i,2})$ and obtain s_i, for $i \in [\kappa]$; it then outputs $s = \sum_{i=1}^{\kappa} 2^{i-1} s_i$. ◀

▶ Remark. We note that it is also possible to use Schnorr's proof (without extractability) for better computational efficiency, but at the cost of one more round. Namely, instead of directly posting the partial public keys on the bulletin board, we let the trustees first post a commitment of their partial public keys, and then decommit them. For instance, we can use simple hash based commitment. To commit m, pick a random $d \leftarrow \{0,1\}^\lambda$, output $c := H(m\|d)$. To verify a commitment, just check if $c = H(m\|d)$. Now the simulator can fix the combined public key to the one that the simulator knows its corresponding secrete key by equivocating the commitments. (cf. [10] for more details of this technique.)

NIZK for knowledge of plaintext. In our scheme, the voters post encryptions of their temporal ID on the BB. In order to prevent the adversary from copying and modifying their temporal ID, we use NIZK for the correctness of $\mathsf{TRE.Enc}$ algorithm as the following.

$$\mathsf{NIZK}_{\mathcal{R}_5}\big\{(\mathtt{pk}, e), (\omega, m) : e = \mathsf{TRE.Enc}(\mathtt{pk}, m; \omega)\big\}$$

With regard to ElGamal encryption, the proof of knowledge of plaintext and randomness is the same as proof of knowledge of randomness, as given r, everyone can compute $m := e_2/\mathtt{pk}^r$. This can be done via strong Fiat-Shamir heuristic on Schnorr's proof [46]. However, this NIZK assume the plaintext m is public. In practice, if the message space is small, we can use Sigma OR-composition to numerate each possible plaintext. However, this is not efficient. Alternatively, we we propose a Sigma protocol for knowledge of plaintext in Fig. 19.

Σ protocol for knowledge of plaintext

Statement: h and $(e_1, e_2) = (g^r, m \cdot h^r)$
Witness: $m \in \mathbb{G}$ and $r \in \mathbb{Z}_p$
Prover:
- Pick random $S \leftarrow \mathbb{G}$ and $t \in \mathbb{Z}_p$.
- Send $(c_1, c_2) = (g^t, S \cdot h^t)$ to the verifier.

Verifier:
- Send random challenge $\rho \leftarrow \{0,1\}^\lambda$ to the prover.

Prover:
- Send $u := r \cdot \rho + t$ and $W := m^\rho \cdot S$ to the verifier.

Verifier:
- Return valid if and only if $e_1^\rho \cdot c_1 = g^u$ and $e_2^\rho \cdot c_2 = W \cdot h^u$.

Figure 19 Σ protocol for knowledge of plaintext.

▶ **Theorem 32.** *The* NIZK *described in Fig. 19 is a Sigma proof of knowledge of $m \in \mathbb{G}$ and $r \in \mathbb{Z}_p$ for $(e_1, e_2) = (g^r, m \cdot h^r)$.*

Proof. Perfect completeness follows by inspection. To special soundness, we can construct a knowledge extractor that takes in two set of valid transcripts $(c_1, c_2, \rho_1, u_1, W_1)$

and $(c_1, c_2, \rho_2, u_2, W_2)$ can output the witness. Indeed, we have $r := \frac{u_1 - u_2}{\rho_1 - \rho_2}$ and $m := (W_1/W_2)^{1/(\rho_1-\rho_2)}$. Finally, for special honest verifier zero-knowledge, we will construct a PPT simulator Sim that given any challenge ρ^* can outputs a valid transcript that is indistinguishable from the real one. The simulator Sim first picks random $u \leftarrow \mathbb{Z}_p$ and $W \leftarrow \mathbb{G}$. It then computes $c_1 := g^u / e_1^{\rho^*}$ and $c_2 := W \cdot h^u / e_2^{\rho^*}$. It is easy to see that (c_1, c_2, ρ^*, u, W) has identical distribution as the real transcript. ◀

One-out-of-many NIZK. In our scheme, the voters need to use

$$\mathsf{NIZK}_{\mathcal{R}_6}\{(\mathtt{pk}, (e_1, \ldots, e_n), e'), (\omega, i) : e' = \mathsf{TRE.ReRand}(\mathtt{pk}, e_i; \omega)\}$$

to show that e' is re-randomized from one of a set of ciphertexts as follows. The statement can be re-stated as to show that one of the ciphertexts $(e_1/e', \ldots, e_n/e')$ is encryption of 0; namely, the prover knows i and r such that $e_i/e' := \mathsf{TRE.Enc}(\mathtt{pk}, 0; r)$. Groth and Kohlweiss [32] proposed an efficient one-out-of-many proof, whose proof size is $O(\log n)$. Their proof is a 3-move public coin special honest verifier zero-knowledge proof that allows the prover to convince the verifier that one out of a set of commitment commits to 0. Although they instantiate their proof to Pedersen commitment, their protocol is also compatible with ElGamal commitment/encryption. Therefore, we can use strong Fiat-Shamir heuristic on their proof to instantiate our $\mathsf{NIZK}_{\mathcal{R}_6}$, and no knowledge extractor is needed. Due to space limitation, we refer interested readers to [32] for more details.

NIZK for shuffle correctness. Each trustee is shuffling the set of *triple ciphertext* (ballot) in turn. We need shuffle NIZK for the correctness of re-encryption mix-net, i.e.,

$$\mathsf{NIZK}_{\mathcal{R}_7}\left\{\begin{array}{c}(\mathtt{pk}, (e_1, \ldots, e_n), (e'_1, \ldots, e'_n)), (\Pi, (\omega_1, \ldots, \omega_n)) : \\ \forall i \in [n] : e'_i = \mathsf{TRE.ReRand}(\mathtt{pk}, e_{\Pi(i)}; \omega_i)\end{array}\right\}.$$

There are many ZK/NIZK of shuffling correctness for ElGamal re-encryption. To our best knowledge, the most efficient one is proposed by Bayer and Groth [7]. The proof size of their ZK is $O(\sqrt{n})$. Although the original proof is for shuffling single ElGamal ciphertexts rather than bundles of three ciphertexts, it is easy to modify their proof to meet our requirement. More concretely, the modified protocol consists of two sub-protocols. Let ρ be the permutation. The prover first uses generalized Pedersen commitment to commit $x^{\rho(1)}, \ldots, x^{\rho(n)}$ and prove its correctness, where x is randomly chosen by the verifier; after that, the prover uses multi-exponentiation argument to show that $\prod_{i=1}^n (e_{i,j})^{x^i} = \mathsf{TRE.Enc}(\mathtt{pk}, 0; s) \cdot \prod_{i=1}^n (e'_{i,j})^{x^{\pi(i)}}$ for $j \in [3]$, where s is some randomness known to the prover. Their protocol is Fiat-Shamir friendly, and we refer interested readers to [7] for more details.

NIZK for share decryption correctness The NIZK proof of membership

$$\mathsf{NIZK}_{\mathcal{R}_8}\{(\overline{\mathtt{pk}}_i, e_1, \overline{m}_i), (\overline{\mathtt{sk}}_i) : \overline{\mathtt{pk}}_i = g^{\overline{\mathtt{sk}}_i} \wedge \overline{m}_i = e_1^{\overline{\mathtt{sk}}_i}\}$$

invoked above can be instantiated by strong Fiat-Shamir heuristic on the well-known Chaum-Pedersen proof [24] for DDH tuples.