Digital Liquid Democracy: How to Vote Your Delegation Statement

Bingsheng Zhang
Lancaster University
b.zhang2@lancaster.ac.uk

Hong-Sheng Zhou
Virginia Commonwealth University
hszhou@vcu.edu

June 24, 2017

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 collective decision making. 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". Very recently, a few liquid democracy implementations, e.g. Google Votes and Decentralized Autonomous Organization (DAO), are released; however, those systems only focus on the functionality aspect, as no privacy/anonymity is considered.

In this work, we, for the first time, provide a rigorous study of liquid democracy under the Universal Composability (UC) framework. In the literature, liquid democracy was achieved via two separate stages – delegation and voting. We propose an efficient liquid democracy e-voting scheme that unifies these two stages. At the core of our design is a new voting concept called *statement voting*, which can be viewed as a natural extension of the conventional voting approaches. 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.

Contents

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. Although the former one treats every voter equally, it is not scalable; therefore, the latter one is widely used in most countries. However, representative democracy 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. What's worse, misinformation campaigns often influence the voters to select certain candidates which could be against the voters' own interests. In the past decades, the concept of liquid democracy [21] 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[1]. We here ask the following challenging question:

Is it possible to introduce new technologies to circumvent the implementation barriers to liquid democracy?

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. In terms of liquid democracy, 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 can also state whether he/she is willing to be delegated in the ballot. To obtain the basic intuition, let's first leave privacy aside and consider the following toy example.

Example: Each ballot is in the form of $(\mathsf{ID}, \mathtt{action}, \mathsf{target})$. If a voter V_i is willing to be delegated, he/she sets $\mathsf{ID} := \mathsf{V}_i$; otherwise, sets $\mathsf{ID} := \perp$. If V_i wants to delegate his/her vote to another voter V_j, then the ballot is $B := (\mathsf{ID}, \mathtt{delegate}, \mathsf{V}_j)$. If V_i wants to directly vote for v_i, then the ballot is $B := (\mathsf{ID}, \mathtt{vote}, v_i)$. Suppose there are seven ballots: $B_1 := (\mathsf{V}_1, \mathtt{delegate}, \mathsf{V}_7)$, $B_2 := (\mathsf{V}_2, \mathtt{vote}, v_2)$, $B_3 := (\mathsf{V}_3, \mathtt{vote}, v_3)$, $B_4 := (\perp, \mathtt{vote}, v_4)$, $B_5 := (\mathsf{V}_5, \mathtt{delegate}, \mathsf{V}_4)$, $B_6 := (\perp, \mathtt{delegate}, \mathsf{V}_3)$ and
$B_7 := (\mathsf{V}_7, \mathtt{delegate}, \mathsf{V}_3)$. 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 v_3; that means, B_7 votes for v_3 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 v_3. 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 $(v_3, v_2, v_3, v_4, \perp, v_3, v_3)$.

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. In "our design" paragraph in the Introduction, we will present the first *digital* construction for implementing full-fledged liquid democracy. How to construct a physical construction is an interesting open question. Before presenting our construction, we first need to clearly define and model our security goal.

[1]All the existing liquid democracy implementations, e.g., Google Votes and Decentralized Autonomous Organization (DAO) do not consider privacy/anonymity. This drawback prevents them from being used in serious elections. Here, we note that straightforword blockchain-based solutions cannot provide good privacy in practice. Although some blockchains such as Zerocash [5] can be viewed as a global mixer, they implicitly require anonymous channels. While 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 can easily identify the user during quiet hours.

Modeling liquid democracy voting. We for the first time provide a rigorous modeling for liquid democracy voting. More concretely, we model liquid democracy voting in the well-known Universal Composability (UC) framework, via an ideal functionality $\mathcal{F}_{\textsc{Liquid}}$. The functionality interacts with a set of voters, trustees, and an auditor, and consists of preparation phase, voting/delegation phase, and tally phase. During the preparation phase, the trustees need to indicate their presence to $\mathcal{F}_{\textsc{Liquid}}$, and the election/voting will not start until all the trustees have participated the preparation. During the voting/delegation phase, each voter can either directly vote for the candidate(s) or delegate his vote to another voter. In addition, each voter can use a flag α to indicate whether he is willing to be delegated. Note that, when all the trustees are corrupted, $\mathcal{F}_{\textsc{Liquid}}$ leaks the voters' ballots to the adversary.

To model the privacy of mixing type of e-voting, we let $\mathcal{F}_{\textsc{Liquid}}$ replace each voter's ID with a pseudonym. The functionality $\mathcal{F}_{\textsc{Liquid}}$ then sorts the anonymized ballots lexicographically to hide the order correspondence and reveal them to the adversary. Note that this type of information leakage is consistent with the conventional paper-based voting system, where a voter submits his ballot to a ballot box and the ballot is mixed together with the other voters' ballots. To model end-to-end verifiability, we introduce an auditor Au who will verify whether the election result is mis-presented. Note that the auditor can be performed by any party, and here we model auditor as a single entity for simplicity.

We emphasize that in practice, virtually all threshold cryptographic systems suffer from selective failure attacks; namely, during the opening process of a threshold cryptographic system, the last several share holders can jointly see the content to be opened themselves before hand. Hence, they can decide if they want to actually open the content. However, surprisingly, this subtle issue was rarely modeled in the literature. For instance, the only previously known e-voting functionality [22] fails to address it. During the tally phase, the tally will be released if all the trustees agree to proceed.

Our design. Our toy example shows that it is possible to interpret the delegation statement by extending the tally algorithm. However, it is not immediately obvious about how to apply the same technique in conjunction with privacy. Before the voting/delegation phase, 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 $B_i := (w_i, \mathtt{delegate}, W_j)$, if he wants to delegate to V_j; or he sets $B_i := (w_i, \mathtt{vote}, v_i)$, if he wants to vote for v_i; 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.

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 remark 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 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.

Related work. The concept of liquid democracy (a.k.a. delegative democracy) is emerging over the last decades [29, 2, 9]. To our best knowledge, Ford [21] 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. Google Votes [24] 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. Decentralized Autonomous Organization (DAO) [20] realized liquid democracy using the blockchain technology, and it has been widely used to vote on expenses and actions of various contracts. Recently, Merkle [28] provide a comprehensive review on DAO, collective intelligence, and liquid democracy. He believes that liquid democracy can utilize the expertise of all the citizens to make high-quality decisions. Nevertheless, 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.

With regards to conventional security oriented e-voting systems, Chaum [16] proposed to use anonymous channels and pseudonyms to achieve voter privacy and verifiability. Benaloh and Yung later showed how to distribute the centralized election authority using threshold cryptography [7]. Sako and Kilian [31] minimized the assumption needed for a mix-type voting system to achieve so-call receipt-freeness, where the voters can simulate/hide their votes to the coercers. Groth [22] gave the first UC definition for an e-voting system, and he proposed a protocol using (threshold) homomorphic encryption. Moran and Naor [30] later studied the privacy and receipt-freeness of an e-voting system in the stand-alone setting. Unruh and Muller-Quade [34] gave a formal study of e-voting coerciability in the UC framework. Alwen *et al.* [3] considered stronger versions of coerciability in the MPC setting under UC framework.

The most widely used e-voting system in practice is Helios [1]. Almost all the end-to-end verifiable e-voting systems [19, 17, 26, 25] requires a consistent bulletin board (BB). However, none of them gives a practical realization of BB. On the other hand, Kiayias, Zhou, and Zikas [27] gives a UC model of the global ledger and discuss about how to use such a ledger to enable fair and robust MPC.

Organization. In Section 2, we present the required preliminaries including a brief overview of UC framework and some useful ideal functionalities. In Section 3, we rigorously model liquid democracy voting, and in Section 4, we present the details of our construction for liquid democracy voting. The instantiations of the building blocks in our main construction can be found in Section 5, and some discussions for future directions can be found in Section 6. Finally, the security analysis of our main construction can be found in in Appendix A.

2 Preliminaries

2.1 The UC framework

Following Canetti's framework [12, 11], 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 $r = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, r_2, \ldots$ 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 2.1. *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.

2.2 Ideal functionalities

2.2.1 Bulletin board functionality

The public bulletin board (BB) is modeled as a global functionality $\bar{\mathcal{G}}_{\text{BB}}$. Formal description can be found in Fig. 1. 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 $\bar{\mathcal{G}}_{\text{BB}}$ can be much simplified version of the global public ledger functionality $\bar{\mathcal{G}}_{\textsc{ledger}}$ recently defined by Kiayias et al [27].

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

The shared functionality $\bar{\mathcal{G}}_{\text{BB}}$ 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 $\mathcal{S}$, 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 $\mathcal{S}$, return $(\textsc{read}, \mathsf{sid}, \mathsf{state})$ to the requestor.

Figure 1: The public bulletin board functionality.

2.2.2 Certificate functionality

We present the multi-session version of certificate functionality following the modeling of [13]. 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. 2.

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

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

- Upon receiving $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, m)$ from a signer $P \in \{\mathsf{S}_1, \ldots, \mathsf{S}_k\}$, 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 $P \in \{\mathsf{R}_1, \ldots, \mathsf{R}_n\}$, 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 2: The multi-session functionality for certificate.

3 Modeling Liquid Democracy Voting

System architecture. Our liquid democracy e-voting system consists of a set of trustees $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$, a set of voters $\mathbb{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, and an auditor Au. Similar to all existing end-to-end verifiable e-voting systems, our system requires a consistent bulletin board, which is modelled as a global functionality $\bar{\mathcal{G}}_{\mathrm{BB}}$. We note that the auditor can be performed by any party; for simplicity, we model auditor as a single entity.

Functionality $\mathcal{F}_{\textsc{Liquid}}$

The functionality $\mathcal{F}_{\textsc{Liquid}}$ 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. $\mathcal{F}_{\textsc{Liquid}}$ is parameterized by an algorithm $\mathsf{TallyProcess}$, a table Tab, and variables $result$, T_1, T_2, T_3, and B_i for all $i \in [n]$.

Initially, set $result := \emptyset$, $T_1 := \emptyset$, $T_2 := \emptyset$, $T_3 := \emptyset$; 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 $\mathcal{S}$.
2. Upon receiving input $(\textsc{InitialVoter}, \mathsf{sid}, \eta)$ from the voter $\mathsf{V}_i \in \mathbb{V}$, if $|T_1| < k$, ignore the input.
   Otherwise, send $(\textsc{InitialVoterNotify}, \mathsf{sid}, \mathsf{V}_i)$ to the adversary $\mathcal{S}$. If $|\mathbb{T}_{\mathsf{corrupt}}| = k$, then additionally send a message $(\textsc{DelLeak}, \mathsf{sid}, \mathsf{V}_i, \eta)$ to the adversary $\mathcal{S}$.
   Upon receiving $(\textsc{VoterID}, \mathsf{sid}, \mathsf{V}_i, w_i)$ from $\mathcal{S}$,
   if $\mathsf{V}_i \in \mathbb{V}_{\mathsf{corrupt}}$, then set $\mathsf{Tab}[i] := w_i$;
   otherwise, if $\mathsf{V}_i \in \mathbb{V}_{\mathsf{honest}}$ and $\eta = 0$, then set $\mathsf{Tab}[i] := \perp$;
   else, generate random temporal ID $w'_i \leftarrow \{0,1\}^\lambda$, and set $\mathsf{Tab}[i] := w'_i$.

Voting/Delegation:

1. Upon receiving input $(\textsc{Delegate}, \mathsf{sid}, \mathsf{V}_j)$ from the voter $\mathsf{V}_i \in \mathbb{V}$, if $|T_1| < k$, ignore the input. Otherwise, record $B_i := (\mathsf{Tab}[i], \mathtt{delegate}, \mathsf{Tab}[j])$; send a message $(\textsc{ExecuteNotify}, \mathsf{sid}, \mathsf{V}_i)$ to the adversary $\mathcal{S}$. If $|\mathbb{T}_{\mathsf{corrupt}}| = k$, then additionally send a message $(\textsc{Leak}, \mathsf{sid}, \mathsf{V}_i, \textsc{Delegate}, \mathsf{V}_j)$ to the adversary $\mathcal{S}$.
2. Upon receiving input $(\textsc{Vote}, \mathsf{sid}, v_i)$ from the the voter $\mathsf{V}_i \in \mathbb{V}$, where $v_i \in \mathcal{V}$ and $\mathcal{V}$ contains all possible votes, if $|T_1| < k$, ignore the input. Otherwise, record $B_i := (\mathsf{Tab}[i], \mathtt{vote}, v_i)$; send a message $(\textsc{ExecuteNotify}, \mathsf{sid}, \mathsf{V}_i)$ to the adversary $\mathcal{S}$. If $|\mathbb{T}_{\mathsf{corrupt}}| = k$, then additionally send a message $(\textsc{Leak}, \mathsf{sid}, \mathsf{V}_i, \textsc{Vote}, v_i)$ to the adversary $\mathcal{S}$.

Tally:

1. Upon receiving input $(\textsc{Mix}, \mathsf{sid})$ from the the trustee $\mathsf{T}_j \in \mathbb{T}$, set $T_2 := T_2 \cup \{\mathsf{T}_j\}$. Send a notification message $(\textsc{MixNotify}, \mathsf{sid}, \mathsf{T}_j)$ to the adversary $\mathcal{S}$. If $|T_2| = k$, do
   - For $i \in [n]$, if B_i is not defined, set $B_i := (\perp, \mathtt{vote}, \perp)$.
   - Sort $(B_1, \ldots, B_n)$ lexicographically to form a new ballot vector $(\tilde{B}_1, \ldots, \tilde{B}_n)$.
2. Upon receiving input $(\textsc{Tally}, \mathsf{sid})$ from the trustee $\mathsf{T}_j \in T_2$.
   Otherwise, set $T_3 := T_3 \cup \{\mathsf{T}_j\}$.
   Send a notification message $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_j)$ to $\mathcal{S}$.
3. If $|T_3 \cap \mathbb{T}_{\mathsf{honest}}| + |\mathbb{T}_{\mathsf{corrupt}}| = k$,
   send $(\textsc{Reveal}, \mathsf{sid}, (\tilde{B}_1, \ldots, \tilde{B}_n))$ to $\mathcal{S}$.
4. If $|T_3| = k$,
   compute $result \leftarrow \mathsf{TallyProcess}((\tilde{B}_1, \ldots, \tilde{B}_n), \mathsf{Tab})$.
5. Upon receiving input $(\textsc{ReadResult}, \mathsf{sid})$ from a voter $\mathsf{V}_i \in \mathbb{V}$, if $result = \emptyset$, ignore the input.
   Else, return $(\textsc{ResultReturn}, \mathsf{sid}, result)$ to V_i.

Figure 3: The voting functionality.

The liquid democracy functionality. The ideal functionality for liquid democracy voting, denoted as $\mathcal{F}_{\textsc{Liquid}}$, is formally described in Fig. 3. This functionality interacts with n voters, k trustees, and an auditor. It consists of three phases – preparation, voting/delegation, and tally.

Preparation phase. During the preparation phase, both trustees and voters need to indicate their presence to $\mathcal{F}_{\textsc{Liquid}}$ by sending $(\textsc{InitialTrustee}, \mathsf{sid})/(\textsc{InitialVoter}, \mathsf{sid}, \eta)$ to it. Voters can be absent, but the election/voting will not start until all the trustees have participated the preparation. Here $\eta \in \{0, 1\}$ is used to indicate whether this voter is willing to be delegated. Note that, in current version, for simplicity, we do not allow a voter to selectively reject the other voters' delegations; however, we note that the functionality can be easily extended.

Voting/Delegation phase. During the voting/delegation phase, each voter can either directly vote for the candidate(s) or delegate his vote to another voter. More specifically, if a voter wants to vote v_i, he submits $(\textsc{Vote}, \mathsf{sid}, v_i)$ to $\mathcal{F}_{\textsc{Liquid}}$; else, if a voter wants delegate his vote to V_j, he submits $(\textsc{Delegate}, \mathsf{sid}, \mathsf{V}_j)$. Due to delegation privacy, the voter only knows the number of ballots that are delegated to her, but she does not know who delegate to her. This is an important property to make coercion resistant and vote selling resistant possible in the context of liquid democracy. We remark that, for simplicity, we consider the following policy: If a voter refuses to be delegated, all the ballots that have been delegated to her will be re-set to blank ballots. If there exists a delegation loop, all the undefined ballots will also be re-set to blank ballots. As briefly mentioned in the Introduction, alternative policies can be enforced in our modeling. $\mathcal{F}_{\textsc{Liquid}}$ maintains a database of all the submitted ballots. When all the trustees are corrupted, $\mathcal{F}_{\textsc{Liquid}}$ leaks the voters' ballots to the adversary.

Functionality $\mathcal{F}_{\textsc{Liquid}}$ hides the correspondence between the ballots and the voters, but leaks statistical data. To model this fact, $\mathcal{F}_{\textsc{Liquid}}$ randomly generates a mapping table Tab that assigns each voter V_i with a temporal ID $\mathsf{Tab}[i]$. $\mathcal{F}_{\textsc{Liquid}}$ also allows the corrupted voters to show their own favourite temporal ID. The functionality replaces all the labels V_i for some $i \in [n]$ in the ballots with the corresponding temporal ID's $\mathsf{Tab}[i]$, and then it sorts the anonymised ballots lexicographically to hide the order information.

TallyProcess

Input: a set of ballots $\mathcal{B} := (B_1, \ldots, B_n)$ and a set of voter labels $\mathcal{L} := (\mathsf{V}_1, \cdots, \mathsf{V}_n)$

Output: the tally result *result*

Label Check:

- Check the uniqueness of each label in $\mathcal{L}$. If there exist $i_1, \ldots, i_t \in [n]$, $i_1 \neq i_2 \neq \cdots \neq i_t$ s.t. $V_{i_1} = V_{i_2} = \cdots = V_{i_t}$, then replace all the labels $V_{i_1}, \ldots, V_{i_t}$ in $\mathcal{B}$ with $\perp$.

Delegation statement interpretation:

- For each ballot $B_i \in \mathcal{B}$, parse B_i in form of either $(\mathsf{V}_i, \mathtt{delegate}, \mathsf{V}_j)$ or $(\mathsf{V}_i, \mathtt{vote}, v_i)$ and do the following:
  - If B_i is in form of $(\mathsf{V}_i, \mathtt{delegate}, \mathsf{V}_j)$ and $\mathsf{V}_j \neq \perp$, try to locate a ballot B_j in form of $(\mathsf{V}_j, \cdot, \cdot)$. If founded, replace $B_i := B_j$; otherwise, if B_i is in form of $(\mathsf{V}_i, \mathtt{delegate}, \perp)$ set $B_i := (\mathsf{V}_i, \mathtt{vote}, \perp)$.
  - Repeat the above step, until B_i is in form of $(\cdot, \mathtt{vote}, \cdot)$. If there is a delegation loop, define $B_i := (\perp, \mathtt{vote}, \perp)$.
- Denote the processed ballot set as $\mathcal{B}' := (B'_1, \ldots, B'_n)$.

Tally computation:

- Compute *result* $\leftarrow \mathsf{TallyAlg}(B'_1, \ldots, B'_n)$, where $\mathsf{TallyAlg}(\cdot)$ is the original tally algorithm and $(\cdot, \mathtt{vote}, \perp)$ is viewed as a blank ballot.
- Return *result*.

Figure 4: The extended tally processing algorithm.

Tally phase. Let $\mathcal{V}$ be a set of valid votes. Let $\mathsf{TallyAlg}$ be a deterministic symmetric election tally function that takes $\mathcal{V}$ and outputs the tally result. The concrete functionality of $\mathsf{TallyAlg}$ depends on the actual election, and we do not have any restriction on $\mathsf{TallyAlg}$. Take a simple 1-out-of-m election where there are m candidates $\mathcal{C} := (C_1, \ldots, C_m)$ as an example. Each vote $v_i \in \mathcal{V}$ is an element in $\mathcal{C}$. The tally result is an m-vector $\mathbb{Z}_+^m$ whose i-th coordinate is equal to the number of times C_i was chosen in $\mathcal{V}$. In the rest of this paper, we use $v_i = \perp$ to indicate *blank* vote, and $\mathsf{TallyAlg}$ should ignore those inputs.

To handle the delegation statement, we extend the tally algorithm $\mathsf{TallyAlg}$ to $\mathsf{TallyProcess}$, which can handle the delegation. As depicted in Fig. 4, $\mathsf{TallyProcess}$ carried out three tasks: (i) label check, ensuring every label is unique; otherwise, they are re-set to $\perp$; (ii) interpret the delegation statement, figuring out each voter's final ballot; and (iii) compute the tally result using the conventional tally algorithm $\mathsf{TallyAlg}$. Each ballots is in form of either $(X, \mathtt{delegate}, \mathsf{V}_j)$ or $(X, \mathtt{vote}, v_i)$, where X is either the corresponding voter's label V_i for some $i \in [n]$ or $\perp$. If the voter allows to be delegated, X is his label (temporal ID); otherwise, X is set to $\perp$. If the voter wants to delegate his vote to V_j, her ballot is $(X, \texttt{delegate}, \mathsf{V}_j)$. If the voter wants to directly vote $v_i \in \mathcal{V}$, her ballot is $(X, \texttt{vote}, v_i)$ where $\mathcal{V}$ is the set of all possible votes defined by the voting committee. To resolve the delegation, the TallyProcess algorithm needs to "follow the chain" of delegation. Namely, if $(\mathsf{V}_i, \texttt{delegate}, \mathsf{V}_j)$, $(\mathsf{V}_j, \texttt{delegate}, \mathsf{V}_k)$ and $(\mathsf{V}_k, \texttt{vote}, v)$, then V_i's effective vote is v. On the other hand, 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, there is no ballot is in form of $(\mathsf{V}_j, \cdot, \cdot)$; therefore, $(\mathsf{V}_i, \texttt{delegate}, \mathsf{V}_j)$ does not point to any ballot that might lead to an effective vote. Hence, we re-set V_i's ballot as $(\mathsf{V}_i, \texttt{vote}, \perp)$, which is a blank ballot.

Let T' be the set of trustees who decided to open the ballots. Let $\mathbb{T}_{\mathsf{honest}}$ and $\mathbb{T}_{\mathsf{corrupt}}$ denote the set of honest and corrupt trustees, respectively. Selective failure attack is modelled by leaking to the adversary all the ballots when $|T' \cap \mathbb{T}_{\mathsf{honest}}| + |\mathbb{T}_{\mathsf{corrupt}}| = k$, i.e., the number of honest trustees who participate the tally process plus the number of corrupted trustees is k.

4 Constructing Liquid Democracy Voting

In this section, we will present a construction for realizing the voting functionality that we defined.

4.1 Building blocks

4.1.1 Re-randomizable threshold PKE

Our construction requires a *re-randomizable* (t, n)*-threshold public key encryption* scheme. In addition, we expect the key generation of scheme is distributed. Next, we extend the notion proposed by Shoup and Genarro [33] and by Boneh *et al.* [10]. More concretely, a re-randomizable (t, n)-threshold public key encryption RTE consists of a tuple of PPT algorithms RTE.DGen, RTE.Enc, RTE.ReRand, RTE.ShareDec, RTE.ShareVer, RTE.Merge, RTE.Combine, and RTE.Combine^{-1}, as follows:

- RTE.DGen: On input of the security parameter 1^λ, (t, n), an index i and random coins ω, outputs the i-th partial public key and secret key pairs $(\overline{\mathtt{pk}}_i, \overline{\mathtt{sk}}_i)$.
- RTE.Merge: On input of (t, n) and a set of partial public keys $\{\overline{\mathtt{pk}}_i\}_{i=1}^n$, outputs the combined public key $\mathtt{pk}$.
- RTE.Enc: On input of the public key $\mathtt{pk}$, a message m and random coins ω, outputs a ciphertext e.
- RTE.ReRand: On input of the public key $\mathtt{pk}$, a ciphertext e and random coins ω, outputs a re-randomized ciphertext e'.
- RTE.ShareDec: On input of the public key $\mathtt{pk}$, the partial secret key $\overline{\mathtt{sk}}_i$ and a ciphertext e, outputs a message share $\overline{m}_i$.
- RTE.ShareVer: On input of the public key $\mathtt{pk}$, the ciphertext e and the message share $\overline{m}_i$, outputs $0/1$.
- RTE.Combine: On input of the public key $\mathtt{pk}$, a ciphertext e and a set of t message shares $\{\overline{m}_{i_1}, \ldots, \overline{m}_{i_t}\}$, outputs the reconstructed message m.
- RTE.Combine^{-1}: On input of the public key $\mathtt{pk}$, a ciphertext e , the original message m, and a set of $(t-1)$ message shares $\{\overline{m}_{i_1}, \ldots, \overline{m}_{i_{t-1}}\}$, outputs the reconstructed message shares $\{\overline{m}'_j\}_{j\in[n]\setminus\{i_1,\ldots,i_{t-1}\}}$

Definition 4.1. *We say* RTE $= \{$RTE.DGen, RTE.Enc, RTE.ReRand, RTE.ShareDec, RTE.ShareVer, RTE.Merge, RTE.Combine, RTE.Combine$^{-1}\}$ *is a* secure *re-randomizable* (t, n)*-threshold public key encryption if the following properties hold:*

- *Correctness: For all valid key generation outputs:* $(\overline{\mathtt{pk}}_i, \overline{\mathtt{sk}}_i) \leftarrow \mathsf{RTE.DGen}(1^\lambda, (t, n), i, \omega_i)$*,* $i \in [n]$*; let* $\mathtt{pk} \leftarrow \mathsf{RTE.Merge}((t, n), \{\overline{\mathtt{pk}}_i\}_{i=1}^n)$*, the following property hold.*
  - *For any ciphertext* e*, if* $\overline{m}_i \leftarrow \mathsf{RTE.ShareDec}(\mathtt{pk}, i, \overline{\mathtt{sk}}_i, e)$*, then* $\mathsf{RTE.ShareVer}(\mathtt{pk}, \overline{\mathtt{pk}}_i, e, \overline{m}_i) = 1$.
  - *If* $e \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, m; \omega)$ *and* $S = \{\overline{m}_1, \ldots, \overline{m}_t\}$ *is a set of decryption shares, where*
  $$\overline{m}_i \leftarrow \mathsf{RTE.ShareDec}(\mathtt{pk}, i, \overline{\mathtt{sk}}_i, e)$$
  for t *distinct partial secret keys, then* $\mathsf{RTE.Combine}(\mathtt{pk}, e, \{\overline{m}_1, \ldots, \overline{m}_t\}) = m$.
- *IND-CPA security: We say that the* (t, n)*-threshold PKE scheme achieves* indistinguishability under plaintext attacks (IND-CPA) *if for any PPT adversary* $\mathcal{A}$ *the following advantage is negligible.*

$\underline{G_{t,n}^{\mathsf{CPA}}(1^\lambda)}$

1. *$\mathcal{A}$ outputs a set $\mathcal{I}_{\mathsf{corr}} \subset \{1,\ldots,n\}$ of $t-1$ corrupted indices.*
2. *For $i = [n]$, run $(\overline{\mathtt{pk}}_i, \overline{\mathtt{sk}}_i) \leftarrow \mathsf{RTE.DGen}(1^\lambda, (t,n), i, \omega_i)$;*
   set $\mathtt{pk} \leftarrow \mathsf{RTE.Merge}((t,n), \{\overline{\mathtt{pk}}_i\}_{i=1}^n)$
3. *$\mathcal{A}(\{\overline{\mathtt{pk}}_i\}_{i\in[n]}, \{\omega_j\}_{j\in\mathcal{I}_{\mathsf{corr}}})$ outputs m_0, m_1 of equal length;*
4. *$b \leftarrow \{0,1\}$; $e \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, m_b; \omega)$;*
5. *$\mathcal{A}(e)$ outputs b^*; It returns 1 if $b = b^*$; else, returns 0.*

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

$$\mathsf{AdvCPA}_{\mathcal{A},t,n}(1^\lambda) = \left|\Pr[G_{t,n}^{\mathsf{CPA}}(1^\lambda) = 1] - \frac{1}{2}\right| \; .$$

- *Unlinkability: We say a re-randomizable (t,n)-threshold PKE scheme to be* unlinkable *if for any* PPT *adversary $\mathcal{A}$ the following advantage is negligible.*

$\underline{G_{t,n}^{\mathsf{Unlink}}(1^\lambda)}$

1. *The same as step 1 of $G_{t,n}^{\mathsf{CPA}}(1^\lambda)$.*
2. *The same as step 2 of $G_{t,n}^{\mathsf{CPA}}(1^\lambda)$.*
3. *$\mathcal{A}(\{\overline{\mathtt{pk}}_i\}_{i\in[n]}, \{\omega_j\}_{j\in\mathcal{I}_{\mathsf{corr}}})$ outputs e_0, e_1;*
4. *$b \leftarrow \{0,1\}$; $e' \leftarrow \mathsf{RTE.ReRand}(\mathtt{pk}, e_b; \omega)$;*
5. *$\mathcal{A}(e')$ outputs b^*; It returns 1 if $b = b^*$; else, returns 0.*

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

$$\mathsf{AdvUnlink}_{\mathcal{A},t,n}(1^\lambda) = \left|\Pr[G_{t,n}^{\mathsf{Unlink}}(1^\lambda) = 1] - \frac{1}{2}\right| \; .$$

- *Decryption consistency: We say that the (t,n)-threshold PKE scheme achieves* decryption consistency *if for any* PPT *adversary $\mathcal{A}$ the following advantage is negligible.*

$\underline{G_{t,n}^{\mathsf{DC}}(1^\lambda)}$

1. *The same as $G_{t,n}^{\mathsf{CPA}}(1^\lambda)$.*
2. *The same as $G_{t,n}^{\mathsf{CPA}}(1^\lambda)$.*
3. *$\mathcal{A}(\{\overline{\mathtt{pk}}_i\}_{i\in[n]}, \{\omega_j\}_{j\in\mathcal{I}_{\mathsf{corr}}})$ outputs two set of decryption shares*
   $S := \{m_0,\ldots,m_t\}$ and $S' := \{m'_0,\ldots,m'_t\}$;
4. *It returns 1 if and only if*
   - *(a) $\forall m \in S \cup S' : \mathsf{RTE.ShareVer}(\mathtt{pk}, \overline{\mathtt{pk}}_i, e, m) = 1$;*
   - *(b) $\mathsf{RTE.Combine}(\mathtt{pk}, e, S) \neq \mathsf{RTE.Combine}(\mathtt{pk}, e, S')$;*

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

$$\mathsf{AdvDC}_{\mathcal{A},t,n}(1^\lambda) = \Pr[G_{t,n}^{\mathsf{DC}}(1^\lambda) = 1] \; .$$

- *Reconstructability: We say that the (t,n)-threshold PKE scheme achieves* reconstructability *if for all m and any* PPT *adversary $\mathcal{A}$ we have*

$$\left|\begin{array}{l} \Pr[\mathcal{A}(\langle \mathtt{pk}, e, m, \{\overline{\mathtt{pk}}_i\}_{i\in[n]}\rangle, \langle \{\overline{m}_i\}_{i\in[n]}\rangle) = 1] \\ \quad - \Pr[\mathcal{A}(\langle \mathtt{pk}, e, m, \{\overline{\mathtt{pk}}_i\}_{i\in[n]}\rangle, \langle \{\overline{m}_j\}_{j\in\{i_1,\ldots,i_{t-1}\}}\}, \{\overline{m}'_j\}_{j\in[n]\setminus\{i_1,\ldots,i_{t-1}\}}\}\rangle) = 1] \end{array}\right| \leq \mathsf{negl}(\lambda)$$

where

- $(\overline{\mathtt{pk}}_i, \overline{\mathtt{sk}}_i) \leftarrow \mathsf{RTE.DGen}(1^\lambda, (t,n), i, \omega_i)$ *for all $i \in [n]$,*
- $\mathtt{pk} \leftarrow \mathsf{RTE.Merge}((t,n), \{\overline{\mathtt{pk}}_i\}_{i=1}^n)$*, and $e \leftarrow$* $\mathsf{RTE.Enc}(\mathtt{pk}, m; \omega)$
- $\overline{m}_i \leftarrow \mathsf{RTE.ShareDec}(\mathtt{pk}, i, \overline{\mathtt{sk}}_i, e)$*, for all $i \in [n]$,*
- $\{\overline{m}'_j\}_{j\in[n]\setminus\{i_1,\ldots,i_{t-1}\}} \leftarrow \mathsf{RTE.Combine}^{-1}(\mathtt{pk}, e, m, \{\overline{m}_{i_1},\ldots,\overline{m}_{i_{t-1}}\})$

4.1.2 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 4.2 (NIZK Proof of Membership in the RO Model). $\mathsf{NIZK}^{RO}_{\mathcal{R}}.\{\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext}\}$ *is an NIZK Proof of Membership scheme for the relation $\mathcal{R}$ if the following properties hold:*

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

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

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

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

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

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

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

Definition 4.3 (NIZK Proof of Knowledge in the RO Model). $\mathsf{NIZK}^{RO}_{\mathcal{R}}.\{\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext}\}$ *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 $\mathcal{A}$,*

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

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. In addition, we introduce several convenient notations below. We use the following notation to denote the NIZK for the knowledge of the secret key $\overline{\mathtt{sk}}$ and randomness ω the correctness of $\mathsf{RTE.DGen}$ algorithm.

$$\mathsf{NIZK}_{\mathcal{R}_1}\big\{(\overline{\mathtt{pk}}), (\omega, \overline{\mathtt{sk}}) : (\overline{\mathtt{pk}}, \overline{\mathtt{sk}}) = \mathsf{RTE.DGen}(1^\lambda, (t, n), i; \omega)\big\}$$

Similarly, we denote the NIZK for the correctness of $\mathsf{RTE.Enc}$ algorithm as the following.

$$\mathsf{NIZK}_{\mathcal{R}_2}\big\{(\mathtt{pk}, e), (\omega, m) : e = \mathsf{RTE.Enc}(\mathtt{pk}, m; \omega)\big\}$$

Moreover, we need a NIZK to show the given ciphertext e' is re-randomized from one of a set of ciphertexts $e_1, \ldots, e_n$ as follows.

$$\mathsf{NIZK}_{\mathcal{R}_3}\big\{(\mathtt{pk}, (e_1, \ldots, e_n), e'), (\omega, i) : e' = \mathsf{RTE.ReRand}(\mathtt{pk}, e_i; \omega)\big\}$$

Finally, let Π be a permutation over $[n]$. We also need a NIZK for the correctness of shuffle re-encryption as follows.

$$\mathsf{NIZK}_{\mathcal{R}_4}\left\{\ (\mathtt{pk}, (e_1, \ldots, e_n), (e'_1, \ldots, e'_n)), (\Pi, (\omega_1, \ldots, \omega_n)) : \forall i \in [n] : e'_i = \mathsf{RTE.ReRand}(\mathtt{pk}, e_{\Pi(i)}; \omega_i)\ \right\}$$

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

4.2 Protocol description

In this section, we formally describe the construction of our protocol. 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, voting/delegation, and tally. For the sake of notation simplicity, we omitted the processes of filtering invalid messages on $\bar{\mathcal{G}}_{\mathrm{BB}}$. In practice, $\bar{\mathcal{G}}_{\mathrm{BB}}$ main contains many messages with invalid signatures, and all those messages should be ignored.

4.2.1 Preparation phase

As depicted in Fig. 5, 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 $(\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) \leftarrow \mathsf{RTE.DGen}(1^\lambda, (k,k), j; \alpha_j)$. It then uses

$$\pi_j^{(1)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_1}\left\{(\overline{\mathtt{pk}}_j, k, j), (\alpha_j, \overline{\mathtt{sk}}) : (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{RTE.DGen}(1^\lambda, (k,k), j; \alpha_j)\right\}$$

to show that this process is executed correct; 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 $(\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) \leftarrow \mathsf{RTE.DGen}(1^\lambda, (k,k), j; \alpha_j)$ where α_j is the fresh randomness, and then compute
$$\pi_j^{(1)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_1}\left\{\ (\overline{\mathtt{pk}}_j, k, j), (\alpha_j, \overline{\mathtt{sk}}_j) : (\overline{\mathtt{pk}}_j, \overline{\mathtt{sk}}_j) = \mathsf{RTE.DGen}(1^\lambda, (k,k), j; \alpha_j)\ \right\}$$
- 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 $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Upon receiving $(\textsc{InitialVoter}, \mathsf{sid}, \eta)$ 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 $\left\{\langle \mathsf{ssid}, (\overline{\mathtt{pk}}_j, \pi_j^{(1)}), \sigma_j^{(1)}\rangle\right\}_{j\in[k]}$ 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 $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Verify}((\overline{\mathtt{pk}}_j, k, j), \pi_j^{(1)}) = 1$ for $j \in [k]$. If any of the checks is invalid, halt.
- Compute and store $\mathtt{pk} \leftarrow \mathsf{RTE.Merge}((k,k), \{\overline{\mathtt{pk}}_j\}_{j=1}^{k})$.
- If $\eta = 0$, set $w_i = \perp$; else, if $\eta = 1$, randomly selects $w_i \leftarrow \{0,1\}^\lambda$ and compute $W_i \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, w_i; \beta_i)$ with fresh randomness β_i together with
$$\pi_i^{(2)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_2}\left\{\ (\mathtt{pk}, W_i), (\beta_i, w_i) : W_i = \mathsf{RTE.Enc}(\mathtt{pk}, w_i; \beta_i)\ \right\}\ .$$
- 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 $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Figure 5: Liquid democracy voting scheme $\Pi_{\textsc{Liquid}}$ in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part I).

After that, each voter V_i, $i \in [n]$ on receiving $(\textsc{InitialVoter}, \mathsf{sid}, \eta)$ 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 $\mathtt{pk} \leftarrow \mathsf{RTE.Merge}((k,k), \{\overline{\mathtt{pk}}_j\}_{j=1}^{k})$. The value of η indicates whether the voter allowed to be delegated or not. If $\eta = 0$, V_i sets $w_i = \perp$; otherwise, she randomly picks $w_i \leftarrow \{0,1\}^\lambda$. Afterwards, V_i then uses the election public key $\mathtt{pk}$ to encrypt w_i as $W_i \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, w_i; \beta_i)$ with fresh randomness β_i. She also computes

$$\pi_i^{(2)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_2}\left\{(\mathtt{pk}, W_i), (\beta_i, w_i) : W_i = \mathsf{RTE.Enc}(\mathtt{pk}, w_i; \beta_i)\right\}$$

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

4.2.2 Voting/Delegation phase

As depicted in Fig. 6, in the voting/delegation phase, each voter V_i, $i \in [n]$ first fetches all the posted encrypted temporal IDs from $\bar{\mathcal{G}}_{\text{BB}}$, and checks their attached NIZK proofs. For any missing or invalid (encrypted) temporal IDs, the voters replace them with $\mathsf{RTE.Enc}(\mathtt{pk}, \perp; 0)$, which is encryption of $\perp$ with trivial randomness. Moreover, the voters also defines $W_0 \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, \perp; 0)$. (For those voters who want to vote themselves, they will re-randomize W_0.)

If a voter V_i wants to vote v_i, she will produce V_i as a re-randomized W_0 and U_i as encryption of v_i. Meanwhile, she also gives a NIZK proof showing that V_i is re-randomized from one of the ciphertexts in $(W_0, \ldots, W_n)$ and another NIZK proof showing U_i is created by her. Denote the corresponding proofs as $\pi_i^{(3)}$ and $\pi_i^{(4)}$, respectively. V_i signs and posts $(U_i, V_i, \pi_i^{(3)}, \pi_i^{(4)})$ to $\bar{\mathcal{G}}_{\text{BB}}$.

If a voter V_i wants to delegate to voter V_j, she will produce V_i as a re-randomized W_j and U_i as encryption of $\perp$. Similarly, she also gives a NIZK proof showing that V_i is re-randomized from one of the ciphertexts in $(W_0, \ldots, W_n)$ and another NIZK proof showing U_i is created by her. Denote the corresponding proofs as $\pi_i^{(3)}$ and $\pi_i^{(4)}$, respectively. V_i signs and posts $(U_i, V_i, \pi_i^{(3)}, \pi_i^{(4)})$ to $\bar{\mathcal{G}}_{\text{BB}}$.

Voting/Delegation:
Upon receiving $(\textsc{Vote}, \mathsf{sid}, v_i)/(\textsc{Delegate}, \mathsf{sid}, \mathsf{V}_j)$ from the environment $\mathcal{Z}$, the voter V_i operates as the follows:

- If $\mathtt{pk}$ is not defined yet, return $(\textsc{Error}, \mathsf{sid})$ to $\mathcal{Z}$ and halt.
  Otherwise, send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\text{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\text{BB}}$.
  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 $W_\ell \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, \perp; 0)$ if W_ℓ is missing or $b_\ell^{(2)} = 0$ or $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\mathtt{pk}, W_\ell), \pi_\ell^{(2)}) = 0$.
  Define $W_0 \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, \perp; 0)$.
- If $(\textsc{Vote}, \mathsf{sid}, v_i)$ compute
  - $V_i \leftarrow \mathsf{RTE.ReRand}(\mathtt{pk}, W_0; \gamma_i)$
    and $\pi_i^{(3)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_3}\left\{ (\mathtt{pk}, (W_0, \ldots, W_n), V_i), (\gamma_i, \ell) : V_i = \mathsf{RTE.ReRand}(\mathtt{pk}, W_\ell; \gamma_i) \right\}$ where $\ell = 0$.
  - $U_i \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, v_i; \delta_i)$ and $\pi_i^{(4)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_2}\left\{ (\mathtt{pk}, U_i), (\delta_i, v_i) : U_i = \mathsf{RTE.Enc}(\mathtt{pk}, v_i; \delta_i) \right\}$.
- If $(\textsc{Delegate}, \mathsf{sid}, \mathsf{V}_j)$, compute
  - $V_i \leftarrow \mathsf{RTE.ReRand}(\mathtt{pk}, W_j; \gamma_i)$
    and $\pi_i^{(3)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_3}\left\{ (\mathtt{pk}, (W_0, \ldots, W_n), V_i), (\gamma_i, \ell) : V_i = \mathsf{RTE.ReRand}(\mathtt{pk}, W_\ell; \gamma_i) \right\}$ where $\ell = j$.
  - $U_i \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, \perp; \delta_i)$ and $\pi_i^{(4)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_2}\left\{ (\mathtt{pk}, U_i), (\delta_i, \perp) : U_i = \mathsf{RTE.Enc}(\mathtt{pk}, \perp; \delta_i) \right\}$.
- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (U_i, V_i, \pi_i^{(3)}, \pi_i^{(4)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and receive $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (U_i, V_i, \pi_i^{(3)}, \pi_i^{(4)}), \sigma_i^{(3)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{V}_i, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (U_i, V_i, \pi_i^{(3)}, \pi_i^{(4)}), \sigma_i^{(3)} \rangle)$ to $\bar{\mathcal{G}}_{\text{BB}}$.

Figure 6: Liquid democracy voting scheme $\Pi_{\textsc{Liquid}}$ in the $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part II).

4.2.3 Tally phase

The tally phase is depicted in Fig. 7. The trustees first fetches (W_i, V_i, U_i) (which is viewed as the submitted ballot for voter V_i) from $\bar{\mathcal{G}}_{\text{BB}}$ and check their attached NIZK proofs. All the invalid ballots will be discard. Let n' be the 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, V_i, U_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 $\mathsf{RTE.ShareDec}$. More specifically, each trustee will sign and post its decryption shares to $\bar{\mathcal{G}}_{\text{BB}}$.

Each voter can then compute the tally result as follows. The voter first fetches all the decryption shares and checks their validity using $\mathsf{RTE.ShareVer}$. Upon success, the voter uses $\mathsf{RTE.Combine}$ to reconstruct the messages. She then use $\mathsf{TallyProcess}$ as described in Fig. 4 to calculate the final tally.

Tally:

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

- 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 $\ell \in [n]$:
    - 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 $\widehat{\mathcal{F}}_{\textsc{Cert}}$;
    - If $\langle \mathsf{ssid}, (U_\ell, V_\ell, \pi_\ell^{(3)}, \pi_\ell^{(4)}), \sigma_\ell^{(3)} \rangle$, is contained in *state*, then send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (U_\ell, V_\ell, \pi_\ell^{(3)}, \pi_\ell^{(4)}), \sigma_\ell^{(3)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (U_\ell, V_\ell, \pi_\ell^{(3)}, \pi_\ell^{(4)}), b_j^{(3)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$;

  Set $i = 0$. For $\ell \in [n]$, define $e_i^{(0)} := (W_\ell, U_\ell, V_\ell)$ and $i = i + 1$ if the following holds:
    - W_ℓ, U_ℓ, V_ℓ exist in *state* and $b_\ell^{(2)} \cdot b_\ell^{(3)} = 1$;
    - $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\mathtt{pk}, W_\ell), \pi_\ell^{(2)}) = 1$;
    - $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Verify}((\mathtt{pk}, (W_0, \ldots, W_n), V_\ell), \pi_\ell^{(3)}) = 1$;
    - $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\mathtt{pk}, U_\ell), \pi_\ell^{(4)}) = 1$;

  Assume $i = n'$ 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 $\mathsf{NIZK}_{\mathcal{R}_4}.\mathsf{Verify}((\mathtt{pk}, (e_1^{(j-2)}, \ldots, e_{n'}^{(j-2)}), (e_1^{(j-1)}, \ldots, e_{n'}^{(j-1)})), \pi_{j-1}^{(5)}).$) T_j randomly picks a permutation Π_j over [n']; For $i \in [n']$, set $e_{i,1}^{(j)} \leftarrow \mathsf{RTE.ReRand}(\mathtt{pk}, e_{\Pi_j(i),1}^{(j-1)}; r_{i,1}^{(j)})$, $e_{i,2}^{(j)} \leftarrow \mathsf{RTE.ReRand}(\mathtt{pk}, e_{\Pi_j(i),2}^{(j-1)}; r_{i,2}^{(j)})$, and $e_{i,3}^{(j)} \leftarrow \mathsf{RTE.ReRand}(\mathtt{pk}, e_{\Pi_j(i),3}^{(j-1)}; r_{i,3}^{(j)})$, where $r_{i,1}^{(j)}, r_{i,2}^{(j)}, r_{i,3}^{(j)}$ are fresh randomness. Compute
$$\pi_j^{(5)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_4} \left\{ \begin{array}{r} \left(\mathtt{pk}, (e_1^{(j-1)}, \ldots, e_{n'}^{(j-1)}), (e_1^{(j)}, \ldots, e_{n'}^{(j)})\right), \left(\Pi_j, (r_{i,1}^{(j)}, r_{i,2}^{(j)}, r_{i,3}^{(j)})_{i\in[n']}\right) \; : \\ \forall i \in [n'] \; : \; e_{i,1}^{(j)} = \mathsf{RTE.ReRand}\left(\mathtt{pk}, e_{\Pi_j(i),1}^{(j-1)}; r_{i,1}^{(j)}\right) \\ \bigwedge e_{i,2}^{(j)} = \mathsf{RTE.ReRand}\left(\mathtt{pk}, e_{\Pi_j(i),2}^{(j-1)}; r_{i,2}^{(j)}\right) \bigwedge e_{i,3}^{(j)} = \mathsf{RTE.ReRand}\left(\mathtt{pk}, e_{\Pi_j(i),3}^{(j-1)}; r_{i,3}^{(j)}\right) \end{array} \right\}$$
- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}))$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and receive $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), \sigma_j^{(4)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, 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 $\bar{\mathcal{G}}_{\mathrm{BB}}$.

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 $\bar{\mathcal{G}}_{\mathrm{BB}}$. 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 $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), \sigma_\ell^{(4)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), b_j^{(4)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; 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 $i \in [n'], t \in [3]$ compute $\overline{m}_{i,t}^{(j)} \leftarrow \mathsf{RTE.ShareDec}(\mathtt{pk}, \overline{\mathtt{sk}}_j, e_{i,t}^{(k)})$.
- Send $(\textsc{Sign}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,t}^{(j)})_{i\in[n'],t\in[3]})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$ and receives $(\textsc{Signature}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,t}^{(j)})_{i\in[n'],t\in[3]}, \sigma_j^{(5)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\textsc{Submit}, \mathsf{sid}, \langle \mathsf{ssid}, (\overline{m}_{i,t}^{(j)})_{i\in[n'],t\in[3]}, \sigma_j^{(5)} \rangle)$ 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 and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$.
  For $j \in [k]$, if $\langle \mathsf{ssid}, (\overline{m}_{i,t}^{(j)})_{i\in[n'],t\in[3]}, \sigma_j^{(5)} \rangle$ is contained in *state*, send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,t}^{(j)})_{i\in[n'],t\in[3]}, \sigma_j^{(5)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,t}^{(j)})_{i\in[n'],t\in[3]}, b_j^{(5)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$. If $\prod_{j=1}^{k} b_j^{(5)} = 1$, for all $j \in [k], i \in [n'], t \in [3]$, check $\mathsf{RTE.ShareVer}(\mathtt{pk}, e_{i,t}^{(k)}, \overline{m}_{i,t}^{(j)}) = 1$. If any of the above checks is invalid, return $(\textsc{Error}, \mathsf{sid})$ to the environment $\mathcal{Z}$ and halt.
- For $i \in [n']$: compute $m_{i,t} \leftarrow \mathsf{RTE.Combine}((k,k), e_{i,t}^{(k)}, \{\overline{m}_{i,t}^{(j)}\}_{j=1}^{k})$, $t \in [3]$; define $B_i := (m_{i,1}, m_{i,2}, m_{i,3})$.
- Calculate election result $\mathit{result} \leftarrow \mathsf{TallyAlg}(\{B_i\}_{i\in[n']}, (m_{i,1})_{i=1}^{n'})$, and return $(\textsc{ReadResultReturn}, \mathsf{sid}, \mathit{result})$ to $\mathcal{Z}$.

Figure 7: Liquid democracy voting scheme $\Pi_{\textsc{Liquid}}$ in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part III).

4.3 Security

We have the following the theorem.

Theorem 4.4. *Protocol Π_{LIQUID} described in Figure 5, Figure 6 and Figure 7 UC-realizes $\mathcal{F}_{\text{LIQUID}}$ 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 A.

5 Instantiations

In this section, we give efficient instantiations to all the building blocks described in Section 4.1. Our instantiations typically take the advantage of random oracle (RO), aiming for practical efficiency.

5.1 Re-randomizable threshold PKE instantiation

We adopt threshold ElGamal encryption as a candidate for the (k,k)-threshold PKE scheme. For any given security parameter 1^λ, we pick a cyclic group $\langle g \rangle = \mathbb{G}$ with prime order q where the DDH assumption holds. The group information is an implicit input of every algorithm.

- $\mathsf{RTE.DGen}(1^\lambda, (k,k), i)$: randomly pick $\overline{\mathsf{sk}}_i \leftarrow \mathbb{Z}_q$ and output $(\overline{\mathsf{pk}}_i := g^{\overline{\mathsf{sk}}_i}, \overline{\mathsf{sk}}_i)$.
- $\mathsf{RTE.Merge}((k,k), \{\overline{\mathsf{pk}}_i\}_{i=1}^k)$: set $h := \prod_{i=1}^k \overline{\mathsf{pk}}_i$ and output $\mathsf{pk} := (h, \overline{\mathsf{pk}}_1, \ldots, \overline{\mathsf{pk}}_k)$.
- $\mathsf{RTE.Enc}(\mathsf{pk}, m)$: randomly picks $r \leftarrow \mathbb{Z}_q$ and output $e := (g^r, m \cdot h^r)$.
- $\mathsf{RTE.ReRand}(\mathsf{pk}, (e_1, e_2))$: randomly picks $s \leftarrow \mathbb{Z}_q$ and output $e' := (g^s \cdot e_1, h^s \cdot e_2)$.
- $\mathsf{RTE.ShareDec}(\mathsf{pk}, \overline{\mathsf{sk}}_i, (e_1, e_2))$: output $\overline{m}_i := (i, \hat{m}_i := e_1^{\overline{\mathsf{sk}}_i}, \pi_i)$, where $\pi_i \leftarrow \mathsf{NIZK}_{\mathcal{R}_5}\big\{(\overline{\mathsf{pk}}_i, e_1, \hat{m}_i), (\overline{\mathsf{sk}}_i) : \overline{\mathsf{pk}}_i = g^{\overline{\mathsf{sk}}_i} \ \wedge \ \overline{m}_i = e_1^{\overline{\mathsf{sk}}_i}\big\}$.
- $\mathsf{RTE.ShareVer}(\mathsf{pk}, (e_1, e_2), (i, \hat{m}_i, \pi_i))$: check and output $\mathsf{NIZK.Verify}(\overline{\mathsf{pk}}_i, e_1, \hat{m}_i, \pi_i)$.
- $\mathsf{RTE.Combine}((k,k), (e_1, e_2), \{(i, \hat{m}_i, \pi)\}_{i=1}^k)$: output $m := e_2 / \prod_{i=1}^k \hat{m}_i$.
- $\mathsf{RTE.Combine}^{-1}(\mathsf{pk}, e, m, \overline{m}_1, \ldots, \overline{m}_{k-1})$: computes $\overline{m}_k := m / \prod_{i=1}^{k-1} \overline{m}_i$ and $\pi_k \leftarrow \mathsf{NIZK}_{\mathcal{R}_5}.\mathsf{Sim}(\overline{\mathsf{pk}}_k, e_1, \overline{m}_k)$; it then outputs $\overline{m}_k := (k, \overline{m}_k, \pi_k)$.

The NIZK proof of membership

$$\mathsf{NIZK}_{\mathcal{R}_5}\big\{(\overline{\mathsf{pk}}_i, e_1, \hat{m}_i), (\overline{\mathsf{sk}}_i) : \overline{\mathsf{pk}}_i = g^{\overline{\mathsf{sk}}_i} \ \wedge \ \overline{m}_i = e_1^{\overline{\mathsf{sk}}_i}\big\}$$

invoked above can be instantiated by strong Fiat-Shamir heuristic on the well-known Chaum-Pedersen proof [18] for DDH tuples.

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 $\mathsf{AdvCPA}_{\mathcal{A},k,k}(1^\lambda) = \mathsf{negl}(\lambda)$ is guaranteed by the IND-CPA security of the underlying ElGamal encryption which is under the DDH assumption. Besides, $\mathsf{AdvUnlink}_{\mathcal{A},k,k}(1^\lambda) = 0$, as each re-randomized ciphertext has the same distribution as a freshly encrypted ciphertext. Moreover, $\mathsf{AdvDC}_{\mathcal{A},k,k}(1^\lambda) = \mathsf{negl}(\lambda)$ is guaranteed by the uniqueness of the valid message share and the soundness of the NIZK proof $\mathsf{NIZK}_{\mathcal{R}_5}$. Finally, in terms of the reconstructibility, it is obvious that we have an efficient $\mathsf{RTE.Combine}^{-1}$ algorithm such that $\overline{m}_k$ is indistinguishable from the honestly generated ones. This is guaranteed by the uniqueness of the final message share and the simulatability of the NIZK proof $\mathsf{NIZK}_{\mathcal{R}_5}$.

5.2 NIZK instantiations

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.,

$$\mathsf{NIZK}_{\mathcal{R}_1}\big\{(\overline{\mathtt{pk}}), (\omega, \overline{\mathtt{sk}}) : (\overline{\mathtt{pk}}, \overline{\mathtt{sk}}) = \mathsf{RTE.DGen}(1^\lambda, (t,n), i; \omega)\big\}$$

In terms of ElGamal encryption, this NIZK can be realized by strong Fiat-Shamir heuristic of the Schnorr's proof [32]. 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. 8, where $H_1 : \{0,1\}^* \mapsto \mathbb{G}$ is a hash function. $\mathsf{NIZK}_{\mathcal{R}_6}$ allows the prover to show an ElGamal ciphertext is encryption of $0/1$ using a Sigma disjunction of Chaum-Pederden Sigma protocol. $\mathsf{NIZK}_{\mathcal{R}_7}$ 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

$$\pi_i \leftarrow \mathsf{NIZK}_{\mathcal{R}_6}\Big\{\ (g, u, e_{i,1}, e_{i,2}), (s_i, r_i) : (e_{i,1} = g^{r_i} \wedge e_{i,2} = u^{r_i})\ \vee (e_{i,1} = g^{r_i} \wedge e_{i,2}/g = u^{r_i})\ \Big\}\ .$$

- 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

$$\phi \leftarrow \mathsf{NIZK}_{\mathcal{R}_7}\big\{(g, u, e_1, e_2), (r) : (e_1 = g^r \wedge e_2/h = u^r)\big\}\ .$$

- 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 $\mathsf{NIZK}_{\mathcal{R}_6}.\mathsf{Verify}\big\{(g, u, e_{i,1}, e_{i,2}), \pi_i\big\}$.
- Check $\mathsf{NIZK}_{\mathcal{R}_7}.\mathsf{Verify}\big\{(g, u, e_1, e_2), \phi\big\}$.

Figure 8: NIZK for Discrete Logarithm

Theorem 5.1. *The* NIZK *described in Fig. 8 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 $\mathsf{NIZK}_{\mathcal{R}_6}$ and $\mathsf{NIZK}_{\mathcal{R}_7}$. For ZK, the simulator generates $(e_{i,1}, e_{i,2})$ as encryption of 0 and computes $\mathsf{NIZK}_{\mathcal{R}_6}$ honestly. It then simulates ϕ using $\mathsf{NIZK}_{\mathcal{R}_7}.\mathsf{Sim}$. 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 5.2. *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. [8] 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 RTE.Enc algorithm as the following.

$$\mathsf{NIZK}_{\mathcal{R}_2}\big\{(\mathtt{pk}, e), (\omega, m) : e = \mathsf{RTE.Enc}(\mathtt{pk}, m; \omega)\big\}$$

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 $m := e_2/\mathrm{pk}^r$. Therefore, we only need to use strong Fiat-Shamir heuristic on Schnorr's proof [32]. Again, such a proof is not extractable in the UC setting; nevertheless, we don't need the extractor in our security proof.

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

$$\mathsf{NIZK}_{\mathcal{R}_3}\big\{(\mathtt{pk},(e_1,\ldots,e_n),e'),(\omega,i):e'=\mathsf{RTE.ReRand}(\mathtt{pk},e_i;\omega)\big\}$$

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 $e_i/e' := \mathsf{RTE.Enc}(\mathtt{pk},0;r)$. Groth and Kohlweiss [23] 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 $\mathsf{NIZK}_{\mathcal{R}_3}$, and no knowledge extractor is needed. Due to space limitation, we refer interested readers to [23] 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.,

$$\mathsf{NIZK}_{\mathcal{R}_4}\left\{\ (\mathtt{pk},(e_1,\ldots,e_n),(e'_1,\ldots,e'_n)),(\Pi,(\omega_1,\ldots,\omega_n)):\forall i\in[n]:e'_i=\mathsf{RTE.ReRand}(\mathtt{pk},e_{\Pi(i)};\omega_i)\ \right\}\ .$$

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 [4]. 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 $\prod_{i=1}^{n}(e_{i,j})^{x^i} = \mathsf{RTE.Enc}(\mathtt{pk},0;s)\cdot\prod_{i=1}^{n}(e'_{i,j})^{x^{\pi(i)}}$ 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 [4] for more details.

6 Discussions

Voting 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. This line 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 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[2] 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 [6]. 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 multiple party computation to generate a new CRS.

[2]Most non-trivial functionalities (including the e-voting functionality) cannot be UC-realized in the plain model [14, 12, 15].

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. 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.

[4] 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.

[5] 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.

[6] 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.

[7] J. C. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters (extended abstract). In J. Y. Halpern, editor, *5th ACM PODC*, pages 52–62. ACM, Aug. 1986.

[8] 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.

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

[10] D. Boneh, X. Boyen, and S. Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In D. Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 226–243. Springer, Heidelberg, Feb. 2006.

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

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

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

[14] 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.

[15] 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.

[16] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[17] 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.

[18] 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.

[19] 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.

[20] DAO. Create a democratic autonomous organization, 2017. https://www.ethereum.org/dao.

[21] B. Ford. Delegative democracy. 2002. http://www.brynosaurus.com/deleg/deleg.pdf.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] R. Merkle. Daos, democracy and governance. Manuscript, 2016. http://merkle.com/papers/DAOdemocracyDraft.pdf.

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

[30] 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.

[31] K. Sako and J. Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In L. C. Guillou and J.-J. Quisquater, editors, *EUROCRYPT'95*, volume 921 of *LNCS*, pages 393–403. Springer, Heidelberg, May 1995.

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

[33] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, 2002.

[34] 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 Proof for Theorem 4.4

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 $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\text{BB}},\widehat{\mathcal{F}}_{\text{CERT}}}_{\Pi_{\text{LIQUID}},\mathcal{A},\mathcal{Z}}$ where the parties $\mathbb{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ and $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$ run protocol Π_{LIQUID} 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 $\mathsf{EXEC}^{\bar{\mathcal{G}}_{\text{BB}}}_{\mathcal{F}_{\text{LIQUID}},\mathcal{S},\mathcal{Z}}$ where the parties interact with functionality $\mathcal{F}_{\text{LIQUID}}$ 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 $\mathcal{F}_{\text{LIQUID}}$ 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 Π_{LIQUID} 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 $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Ext}^{\mathsf{RO}}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corresponding secret key $\overline{\mathtt{sk}}_j$.
  - Upon receiving $(\textsc{InitialVoterNotify}, \mathsf{sid}, \mathsf{V}_i)$ from the external $\mathcal{F}_{\text{LIQUID}}$ 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 Π_{LIQUID} as if V_i receives $(\textsc{InitialVoter}, \mathsf{sid}, 1)$ from $\mathcal{Z}$.
  - The simulator $\mathcal{S}$ monitoring $\bar{\mathcal{G}}_{\text{BB}}$; 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. The simulator $\mathcal{S}$ acts as V_i to send $(\textsc{InitialVoter}, \mathsf{sid}, 0)$ to $\mathcal{F}_{\text{LIQUID}}$ if $w_i = \perp$; send $(\textsc{InitialVoter}, \mathsf{sid}, 1)$ to $\mathcal{F}_{\text{LIQUID}}$, otherwise. Upon receiving $(\textsc{InitialVoterNotify}, \mathsf{sid}, \mathsf{V}_i)$ from $\mathcal{F}_{\text{LIQUID}}$, the simulator $\mathcal{S}$ sends $(\textsc{VoterID}, \mathsf{sid}, \mathsf{V}_i, w_i)$ to $\mathcal{F}_{\text{LIQUID}}$. Denote the vector of temporal ID as $\mathcal{W} := (w_1, \ldots, w_n)$.
- In the voting/delegation phase:
  - Upon receiving $(\textsc{ExecuteNotify}, \mathsf{sid}, \mathsf{V}_i)$ from the external $\mathcal{F}_{\text{LIQUID}}$ for an honest voter $\mathsf{V}_i \in \mathbb{V} \setminus \mathbb{V}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ creates $V_i \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, 0)$ and $U_i \leftarrow \mathsf{RTE.Enc}(\mathtt{pk}, 0)$. It then simulates the corresponding proofs $\pi_i^{(3)}$ and $\pi_i^{(4)}$. The simulator $\mathcal{S}$ then follows the protocol to post $(V_i, U_i, \pi_i^{(3)}, \pi_i^{(4)})$ to $\bar{\mathcal{G}}_{\text{BB}}$.
  - Monitoring $\bar{\mathcal{G}}_{\text{BB}}$, when a valid $(V_i, U_i, \pi_i^{(3)}, \pi_i^{(4)})$ 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 V_i to w and U_i to v_i. If $w \in \mathcal{W}$, i.e. $w = w_j$ for some $j \in [n]$, the simulator $\mathcal{S}$ acts as V_i to send $(\textsc{Delegate}, \mathsf{sid}, \mathsf{V}_j)$; otheriwse, it acts as V_i to send $(\textsc{Vote}, \mathsf{sid}, v_i)$ to $\mathcal{F}_{\text{LIQUID}}$ to $\mathcal{F}_{\text{LIQUID}}$.

- In the tally phase:
  - Upon receiving $(\textsc{MixNotify}, \mathsf{sid}, \mathsf{T}_j)$ from the external $\mathcal{F}_{\textsc{Liquid}}$ 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 $\Pi_{\textsc{Liquid}}$ as if T_j receives $(\textsc{Mix}, \mathsf{sid})$ from $\mathcal{Z}$.
  - The simulator $\mathcal{S}$ monitoring $\bar{\mathcal{G}}_{\mathrm{BB}}$; once $(e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}$ is posted from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j to send $(\textsc{Mix}, \mathsf{sid})$ to $\mathcal{F}_{\textsc{Liquid}}$.
  - Upon receiving $(\textsc{TallyNotify}, \mathsf{sid}, \mathsf{T}_j)$ from the external $\mathcal{F}_{\textsc{Liquid}}$ for an honest trustee $\mathsf{T}_j \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{corrupt}}$, if $\overline{m}_{i,t}^{(j)}$ are not defined yet, the $\mathcal{S}$ acts as T_j, following the protocol $\Pi_{\textsc{Liquid}}$ as if T_j receives $(\textsc{Tally}, \mathsf{sid})$ from $\mathcal{Z}$.
  - Upon receiving $(\textsc{Reveal}, \mathsf{sid}, (\tilde{B}_1, \ldots, \tilde{B}_n))$ from the external $\mathcal{F}_{\textsc{Liquid}}$, the simulator $\mathcal{S}$ uses the extracted secret key $\overline{\mathtt{sk}}_j$ to compute $\overline{m}_{i,t}^{(j)} \leftarrow \mathsf{RTE.ShareDec}(\overline{\mathtt{sk}}_j, e_{i,t}^{(k)})$ for all the corrupted trustees $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$. The simulator $\mathcal{S}$ then uses $\mathsf{RTE.Combine}^{-1}$ to compute the message shares of the rest honest T_ℓ's message shares $\overline{m}_{i,t}^{(\ell)}$ 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 $\mathsf{EXEC}_{\Pi_{\textsc{Liquid}}, \mathcal{A}, \mathcal{Z}}^{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}}$.
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 A.1. *$\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable.*

Proof. According to Def. 4.3, 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 $\mathsf{RTE.Combine}^{-1}$ instead of using $\mathsf{RTE.ShareDec}$.

Claim A.2. *$\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable.*

Proof. By the decryption consistency property, $\mathsf{AdvDC}_{\mathcal{A},k,k}(1^\lambda) = \mathsf{negl}(\lambda)$, i.e. there is a negligible probability that the adversary can produce another set of valid message shares such that the reconstructed message is different. Since the distribution of the message shares output by $\mathsf{RTE.Combine}^{-1}$ have identical distribution to the real ones, 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 $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Sim}$ to simulate $\pi_i^{(3)}$ and uses $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Sim}$ to simulate $\pi_i^{(4)}$ for all the honest voter $\mathsf{V}_i \in \mathbb{V}$.

Claim A.3. *$\mathcal{H}_3$ and $\mathcal{H}_2$ are indistinguishable.*

Proof. The advantage of the adversary is bounded by the ZK property of NIZK as defined by Def. 4.2. □

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

Claim A.4. *$\mathcal{H}_4$ 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 $\mathsf{AdvCPA}_{\mathcal{A},k,k}(1^\lambda)$ and $\mathsf{AdvUnlink}_{\mathcal{A},k,k}(1^\lambda)$ which are negligible. □

The adversary's view of $\mathcal{H}_4$ is identical to the simulated view $\mathsf{EXEC}_{\mathcal{F}_{\textsc{Liquid}}, \mathcal{S}, \mathcal{Z}}^{\bar{\mathcal{G}}_{\mathrm{BB}}}$. 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 $\widehat{\mathcal{F}}_{\textsc{Cert}}$. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- In the preparation phase:
  - 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 $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Ext}(\overline{\mathtt{pk}}_j, \pi_j^{(1)})$ to extract the corresponding secret key $\overline{\mathtt{sk}}_j$.
  - Upon receiving $(\textsc{DelLeak}, \mathsf{sid}, \mathsf{V}_i, \eta)$ from the external $\mathcal{F}_{\textsc{Liquid}}$ 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 $\Pi_{\textsc{Liquid}}$ as if V_i receives $(\textsc{InitialVoter}, \mathsf{sid}, \eta)$ from $\mathcal{Z}$.
- In the voting/delegation phase:
  - Upon receiving $(\text{LEAK}, \mathsf{sid}, \textsc{Delegate}, \mathsf{V}_j)$ or $(\textsc{Leak}, \mathsf{sid}, \mathsf{V}_i, \textsc{Vote}, v_i)$ from the external $\mathcal{F}_{\textsc{Liquid}}$ 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 $\Pi_{\textsc{Liquid}}$ as if V_i receives $(\textsc{Delegate}, \mathsf{sid}, \mathsf{V}_j)$ or $(\textsc{Vote}, \mathsf{sid}, v_i)$ from $\mathcal{Z}$.
  - Monitoring $\bar{\mathcal{G}}_{\text{BB}}$, when a valid $(V_i, U_i, \pi_i^{(3)}, \pi_i^{(4)})$ 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 V_i to w_j for some $j \in [0, n]$ and U_i to v_i. If $w_j = \perp$, the simulator $\mathcal{S}$ acts as V_i to send $(\textsc{Vote}, \mathsf{sid}, v_i)$ to $\mathcal{F}_{\textsc{Liquid}}$; otheriwse, it acts as V_i to send $(\textsc{Delegate}, \mathsf{sid}, \mathsf{V}_j)$ to $\mathcal{F}_{\textsc{Liquid}}$.
- In the tally phase:
  - The simulator $\mathcal{S}$ monitoring $\bar{\mathcal{G}}_{\text{BB}}$; once a $(e_i^{(j)})_{i=1}^{n'}, \pi_j^{(5)}$ is posted from a corrupted trustee $\mathsf{T}_j \in \mathbb{T}_{\mathsf{corrupt}}$, the simulator $\mathcal{S}$ acts as T_j to send $(\textsc{Mix}, \mathsf{sid})$ to $\mathcal{F}_{\textsc{Liquid}}$.

Indistinguishability. The indistinguishability in this case is straightforward, as $\mathcal{S}$ never simulate a single message to either any corrupted parties or the external $\bar{\mathcal{G}}_{\text{BB}}$. The simulator $\mathcal{S}$ knows all the honest voters' ballot from the external $\mathcal{F}_{\textsc{Liquid}}$, it simply acts as the honest voters according to the protocol $\Pi_{\textsc{Liquid}}$. 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 $\mathcal{F}_{\textsc{Liquid}}$ 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 $\Pi_{\textsc{Liquid}}$.

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

□