Statement Voting

Bingsheng Zhang[1(✉)] and Hong-Sheng Zhou[2]

[1] Lancaster University, Bailrigg, UK
b.zhang2@lancaster.ac.uk
[2] Virginia Commonwealth University, Richmond, USA
hszhou@vcu.edu

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

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

1 Introduction

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

I. Goldberg and T. Moore (Eds.): FC 2019, LNCS 11598, pp. 667–685, 2019.
https://doi.org/10.1007/978-3-030-32101-7_38

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

A New Concept. We could approach the above problem via multiple angles. In this paper, we propose a new powerful 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/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. More specifically, in a statement voting, the ballots typically contain a conditional statement that requires external inputs (a.k.a. parameters and/or arguments) to be executed. For simplicity of illustration, here we consider (nested) if-statements or switch-statements: `If` A `and` B `then` C_1 `else` C_2, where A, B are conditions and C_1, C_2 are election candidates. We emphasize that A and B are usually not defined yet at the time this ballot is created; In the case that, A and B are defined, i.e., all the necessary information is readily collected during the voting phase, the voter can evaluate such a statement himself, and statement voting boils down to conventional voting. Thus, statement voting can be viewed as a non-trivial extension of conventional voting. We note that statement voting can be very flexible. For instance, a ballot statement could be "if tomorrow is rainy, I vote for 'staying at home'; otherwise, I vote for 'hiking'." Note that the ballot can be cast today without even being aware of tomorrow's weather.

Single Transferable Vote (STV) is a special case of statement voting, where the voters rank the election candidates instead of naming only one candidate in their ballots. The ranked candidate list together with the STV tally rule can be viewed as an outcome-dependent statement. Roughly speaking, the statement declares that if my favorite candidate has already won or has no chance to win, then I would like to vote for my second favorite candidate, and so on[1].

Modeling Statement Voting. We provide a rigorous modeling for statement voting. More concretely, we model statement voting in the well-known Universal Composability (UC) framework, via an ideal functionality $\mathcal{F}_{\mathrm{SV}}$. The functionality interacts with voters and trustees, where trustees are the set of voting committee members who prepare the election and calculate the tally result. In our formulation, we introduce a *family of functionalities* to facilitate various realizations. In practice, there is a trade-off between efficiency and privacy guarantees; typically, more efficient constructions yield more privacy leakage. To capture various leakage scenarios, in our ideal functionality, a working table $\mathbb{W}$ is introduced to trace the election transcripts. Depending on which parties are corrupted (and which scheme is considered), some part of the working table will be leaked to the adversary.

Realizing Statement Voting. In this work, we provide several methods to implement statement voting. Similar to most conventional e-voting systems, we

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

assume a trusted *Registration Authority* (RA) to ensure voter eligibility and a consistent *Bulletin Board* (BB) where the voting transactions and result will be posted. The protocol involves a set of voters and a set of trustees, where the trustees are the set of voting committee members who prepare the election and compute the tally.

A Fully Homomorphic Encryption (FHE) Based Scheme. Intuitively, in this scheme, the trustees first run a distributed key generation protocol to setup the voting public key PK. Each voter V_i then encrypts, signs and submits their *voting statements*, x_i (in forms of $(\mathsf{PID}_i, \mathsf{Enc}_{\mathrm{PK}}(x_i))$) to the BB. To prevent re-play attacks, non-interactive zero-knowledge (NIZK) proofs are necessary to ensure the voter knows the plaintext included in his/her submitted ciphertext. After that, the tally processing circuit is evaluated over $\{(\mathsf{PID}_i, \mathsf{Enc}_{\mathrm{PK}}(x_i))\}_{i\in[n]}$ by every trustee. The final tally ciphertext is then decrypted by the trustees and the result will be announced on the BB.

A Publicly Auditable MPC Based Scheme. Intuitively, we can adopt BDO-type of publicly auditable MPC [4], where the trustees form the MPC system. During the preparation phase, they pre-compute sufficiently many correlated randomness (e.g., Beaver triples), and also set up a voting public key. Each voter V_i then encrypts, signs and submits their *voting statements*, x_i (in forms of $(\mathsf{PID}_i, \mathsf{Enc}_{\mathrm{PK}}(x_i))$) together with necessary NIZK proofs to the BB. After that, the trustees perform MPC online computation to first decrypt those encrypted ballots and then evaluate the tally processing circuit over the secretly shared ballots. Finally, the tally result will be posted on the BB. Note that during the online phase, the BDO MPC scheme also posts audit information on the BB to enable public verifiability.

Application: Liquid Democracy. In the past decades, the concept of liquid democracy [16] 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 provably secure solution in the form of either paper-voting or e-voting yet. Liquid democracy can be viewed as a special case of statement voting. 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. Therefore, we can have an immediate construction for liquid democracy based on the above FHE-based and MPC-based schemes. In addition to those "generic" constructions, we also show how to realize liquid democracy with a more efficient construction. In Sect. 3.1, we first define an ideal functionality for liquid democracy, and we then provide a mix-net based construction. Note that the tally processing function must be symmetric, otherwise we cannot use mix-net.

Further Remarks. In this work, we initiate the study of statement voting and liquid democracy. 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. 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? 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. On the other hand, if we don't allow delegation loop in a liquid democracy voting, to what extend can we guarantee voter privacy? How to refine the conventional e-voting privacy to fit liquid democracy is still an open problem. We emphasize that, voting policies can be heavily influenced by local legal and societal conditions. How to define "right" voting policy itself is a very interesting question. We believe our techniques have the potential to help people to identify suitable voting policies which can further eliminate the barriers to democracy. Finally, we note that several important security requirements, e.g., coercion resilience, have not been investigated in this work. See more details in Sect. 4.

Related Work. To our best knowledge, Ford [16] first officially summarized the main characteristics of liquid democracy and brought it to the vision of computer science community. However, in terms of implementation/prototyping, there was no system that can enable liquid democracy until very recently. All the existing liquid democracy voting systems only focus on the functionality aspect of liquid democracy, and no privacy or some other advanced security properties were considered. For instance, Google Votes [20] 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. Similarly, systems such as LiquidFeedback [26], Adhocracy [1], GetOpinionated, [15] also fail to offer provable security guarantees. It is worth mentioning that Sovereign [29] is a blockchain-based voting protocol for liquid democracy; therefore, its privacy is inherited from the underlying blockchain. As a special case of liquid democracy, Kulyk *et al.* proposed several proxy voting schemes [23–25]. In terms of UC modeling on e-voting. Groth [18] gave the first UC definition for an e-voting system, and he proposed a protocol using (threshold) homomorphic encryption. Moran and Naor [27] later studied the privacy and receipt-freeness of an e-voting system in the stand-alone setting. Unruh and Muller-Quade [30] 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. Almost all the end-to-end verifiable e-voting systems [2,13,21,22] requires a consistent bulletin board. Finally, our temporary ID matching technique is closely related to the queried term matching technique used in UnLynx[17] and the anonymous ID linking technique used in [31].

2 Modeling

The parties involved in a statement voting system are a set of trustees $\mathbb{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$, and a set of voters $\mathbb{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$.

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

The functionality $\mathcal{F}_{\mathrm{SV}}$ interacts with voters $\mathbb{V}$, trustees $\mathbb{T}$, and the adversary $\mathcal{S}$. It is parameterized by an algorithm $\mathsf{TallyProcess}$ (see Fig. 2), a working table $\mathbb{W}$, and variables $result$, T_1, T_2, and B_i for all $i \in [n]$. Let $\mathbb{V}_{\mathsf{honest}}$, $\mathbb{V}_{\mathsf{corrupt}}$ and $\mathbb{T}_{\mathsf{honest}}$, $\mathbb{T}_{\mathsf{corrupt}}$ denote the set of honest/corrupt voters and trustees, respectively.

Initially, set $result := \emptyset$, $T_1 := \emptyset$, $T_2 := \emptyset$; for $i \in [n]$, set $B_i := \emptyset$.
Table $\mathbb{W}$ consists of n entries, and each entry consists of voter's real ID, voter's alternative ID, and the statement that the voter submitted; for all $i \in [n]$, the ith entry $\mathbb{W}[i] := (\mathsf{V}_i, w_i, statement_i)$, where $w_i \leftarrow \{0,1\}^{\lambda}$, $statement_i := \emptyset$.

Preparation:

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

Ballot Casting:

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

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

Tally:

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

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

Fig. 1. The voting functionality $\mathcal{F}_{\mathrm{SV}}$.

The Statement Voting Functionality. The ideal functionality for statement voting, denoted as $\mathcal{F}_{\mathrm{SV}}$, is formally described in Fig. 1. 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. $\mathcal{F}_{\mathrm{SV}}$ consists of three phases—Preparation, Ballot Casting, and Tally. The functionality uses a working table $\mathbb{W}$ to track the voters' behavior during the entire ideal execution. The working table $\mathbb{W}$ stores each voter's information including the voter's original ID, his alternative/temporary ID, and the voting statement that he submitted.

Preparation Phase. During the preparation phase, the trustees needs to indicate their presence to $\mathcal{F}_{\mathrm{SV}}$ by sending $(\textsc{InitialTrustee}, \mathsf{sid})$ to it. The election will not start until all the trustees have participated in the preparation.

Ballot Casting Phase. During the ballot casting phase, each voter can submit his voting statement, and this voting statement will be recorded in the corresponding entry. If a voter is corrupt, then he is also allowed to revise his own alternative/temporary ID in the working table. More concretely, based on the input $(\textsc{Cast}, \mathsf{sid}, (s_i, w_i^*))$ from voter V_i, the corresponding entry will be

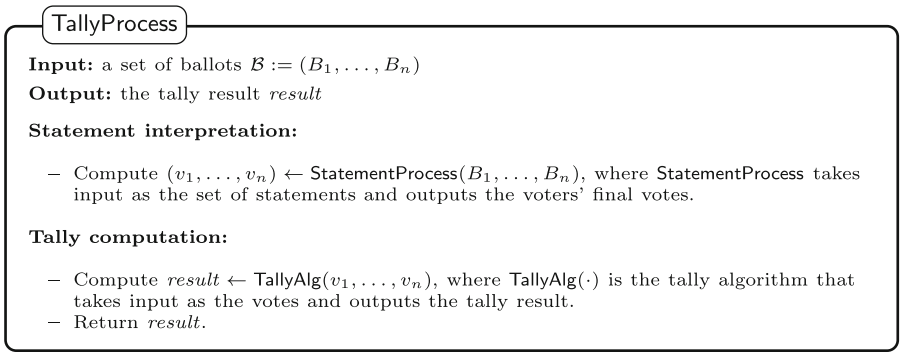

Fig. 2. The extended tally processing algorithm.

updated, i.e., $\mathbb{W}[i] := (\mathsf{V}_i, w_i, s_i)$ if the voter is honest, and $\mathbb{W}[i] := (\mathsf{V}_i, w_i^*, s_i)$ if V_i is corrupt. When all the trustees are corrupted, the functionality $\mathcal{F}_{\mathrm{SV}}$ leaks the entire working tape of the election transcript (i.e., $\mathbb{W}$), to the adversary.

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

The Liquid Democracy Ideal Functionality. Given that liquid democracy is the special case of statement voting, we can easily derive an ideal functionality for liquid democracy from $\mathcal{F}_{\mathrm{SV}}$. The full description of the concrete functionality for liquid democracy, $\mathcal{F}_{\mathrm{LIQUID}}$, can be found in the full version. At a high level, $\mathcal{F}_{\mathrm{LIQUID}}$ uses the following statement interpretation step in the TallyProcess. Each ballot is in form of either $B_i = (w_i, u_i, \perp)$ or $B_i = (w_i, \perp, x_i)$, where w_i and u_i are temporary ID's, and x_i is a vote. To resolve the delegation, the algorithm needs to follow the "chain of delegation", i.e., for each ballot B_i:

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

In case of delegation loop, we set the ballot to blank ballot. Of course, we can enrich the statement by adding another variable to indicate whether a voter wants to be delegated. When the "chain of delegation" breaks by V_i wants to delegate his vote to V_j, while V_j does not want to be delegated. In this case,

V_i's ballot will be re-set to a blank ballot. The most preferable statement for liquid democracy in practice shall be determined by computational social choice theory, which is outside the scope of this paper.

3 Constructions

Due to space limitation, we present the two generic constructions – (i) FHE-based construction and (ii) MPC-based construction, in the full version. In the former one, the voters use FHE to encrypt and upload their statements to the BB. The tally evaluation circuit can be then publicly evaluated over the encrypted statements by any party. After that the trustees will jointly decrypt the final ciphertext(s). In the latter one, any public key encryption scheme can be adopted, so it is more efficient. Similarly, during the voting, the voters encrypt their statements and post them on the BB. The trustees will then participate the MPC evaluation to jointly decrypt the submitted statements and then compute the tally algorithm in the shared format with privacy assurance.

3.1 A Practical Construction for Liquid Democracy

The construction is based on mix-net, and the privacy that it achieves is known as pseudonymity. We emphasize that this level of privacy has been widely accepted and is consistent with the existing paper-based voting systems.

As mentioned before, *liquid democracy* is an emerging type of voting system that receives high attentions since the spread of its concept; however, there is no provably secure solution in the form of either paper-voting or e-voting yet.[2] We now show that how to define a simple statement to enable liquid democracy.

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

Toy Example. Take the Yes/No election as an example. Suppose there are 7 ballots: $B_1 := (\mathsf{V}_1, \mathsf{V}_7, \texttt{delegate})$, $B_2 := (\mathsf{V}_2, \perp, \texttt{vote Yes})$, $B_3 := (\mathsf{V}_3, \perp, \texttt{vote No})$,

[2] All the existing liquid democracy implementations do not consider privacy/anonymity. This drawback prevents them from being used in serious elections. Here, we note that straightforward blockchain-based solutions cannot provide good privacy in practice. Although some blockchains (e.g., Zerocash [5]) can be viewed as a global mixer, they implicitly require anonymous channels. In practice, all the implementations of anonymous channels suffer from time leakage, i.e., the user's ID is only hidden among the other users who are also using the system at the same time. Subsequently, the adversary may easily identify the users during quiet hours.

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

Intuition. At the beginning of each election, the voters V_i, $i \in [n]$, are assigned with a temporary random ID, denoted as ID_i. Let $\mathcal{I} := \{\mathsf{ID}_1, \ldots, \mathsf{ID}_n\}$ be the set of all the voter's random IDs. The voter's statement takes the input as an ID in $\mathcal{I}$, and use it as a reference to point to the corresponding ballot that will be involved in the statement execution, i.e., the potential vote delegation of liquid democracy. To ensure privacy, the voters cannot post their temporary IDs publicly on the bulletin board $\bar{\mathcal{G}}_{\mathrm{BB}}$; however, the voters should be allowed to freely refer to any voter's ID.

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

Building Blocks. Our protocol utilises a bulletin board functionality, a certificate functionality, a threshold re-randomizable encryption scheme, and the corresponding non-interactive zero-knowledge proofs. Their formal descriptions and defintions can be found in the full version.

Bulletin Board Functionality. The public bulletin board (BB) is modeled as a global functionality $\bar{\mathcal{G}}_{\mathrm{BB}}$. 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.

Certificate Functionality. We adopt the multi-session version of certificate functionality following the modeling of [7]. The multi-session certificate functionality $\widehat{\mathcal{F}}_{\mathrm{CERT}}$ 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 to the signers' identities.

Threshold Re-randomizable Encryption. A threshold re-randomizable encryption scheme TRE consists of a tuple of algorithms: $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CombinePK}, \mathsf{CombineSK}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{ReRand})$ as follows.

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

In Appendix A, we provide the corresponding TRE security definitions.

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

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

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

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

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

Preparation

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

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

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

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

Fig. 3. Mix-net based liquid democracy scheme $\Pi_{\textsc{mix-Liquid}}$ in $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part I)

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

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

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

Definition 2 (NIZK PoK in the RO Model). $\mathsf{NIZK}^{\mathrm{RO}}_{\mathcal{R}}.\{\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext}\}$ *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. For all* PPT *adversary* $\mathcal{A}$*, the following is* $1 - \mathsf{negl}(\lambda)$*.*

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

Protocol Description. The protocol is designed in the $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world and it consists of three phases: preparation, ballot casting, and tally. For the sake of notation simplicity, we omit the processes of filtering invalid messages on $\bar{\mathcal{G}}_{\mathrm{BB}}$. In practice, $\bar{\mathcal{G}}_{\mathrm{BB}}$ contains many messages with invalid signatures, and all those messages should be ignored. We will use threshold re-randomizable encryption (TRE) as a building block.

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

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

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

Ballot Casting

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

∘ Round 1:

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

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

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

∘ Round 2:

- Send $(\textsc{Read}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\mathrm{BB}}$, and obtain $(\textsc{Read}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\mathrm{BB}}$. For $\ell \in [n]$, if $\langle \mathsf{ssid}, (W_\ell, \pi_\ell^{(2)}), \sigma_\ell^{(2)}\rangle$ is contained in state, then send $(\textsc{Verify}, \mathsf{sid}, \mathsf{ssid}, (W_\ell, \pi_\ell^{(2)}), \sigma_\ell^{(2)})$ to $\widehat{\mathcal{F}}_{\textsc{Cert}}$, and receive $(\textsc{Verified}, \mathsf{sid}, \mathsf{ssid}, (W_\ell, \pi_\ell^{(2)}), b_j^{(2)})$ from $\widehat{\mathcal{F}}_{\textsc{Cert}}$; For $\ell \in [n]$, set $W_\ell \leftarrow \mathsf{TRE.Enc}(\mathsf{pk}, \perp; 0)$ if W_ℓ is missing or $b_\ell^{(2)} = 0$ or $\mathsf{NIZK}_{\mathcal{R}_5}.\mathsf{Verify}((\mathsf{pk}, W_\ell), \pi_\ell^{(2)}) = 0$.
- (i) If $s_i = (\perp, v_i)$: compute
  - $V_i \leftarrow \mathsf{TRE.ReRand}(\mathsf{pk}, W_0; \gamma_i)$ and $\pi_i^{(3)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_6}\left\{ (\mathsf{pk}, (W_0, \ldots, W_n), V_i), (\gamma_i, 0) : V_i = \mathsf{TRE.ReRand}(\mathsf{pk}, W_\ell; \gamma_i) \right\}$.
  - $U_i \leftarrow \mathsf{TRE.Enc}(\mathsf{pk}, v_i; \delta_i)$ and $\pi_i^{(4)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_5}\left\{ (\mathsf{pk}, U_i), (\delta_i, v_i) : U_i = \mathsf{TRE.Enc}(\mathsf{pk}, v_i; \delta_i) \right\}$.
- (ii) If $s_i = (\mathsf{V}_j, \perp)$: compute
  - $V_i \leftarrow \mathsf{TRE.ReRand}(\mathsf{pk}, W_j; \gamma_i)$ and $\pi_i^{(3)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_6}\left\{ (\mathsf{pk}, (W_0, \ldots, W_n), V_i), (\gamma_i, j) : V_i = \mathsf{TRE.ReRand}(\mathsf{pk}, W_\ell; \gamma_i) \right\}$.
  - $U_i \leftarrow \mathsf{TRE.Enc}(\mathsf{pk}, \perp; \delta_i)$ and $\pi_i^{(4)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_5}\left\{ (\mathsf{pk}, U_i), (\delta_i, \perp) : U_i = \mathsf{TRE.Enc}(\mathsf{pk}, \perp; \delta_i) \right\}$.
- 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 $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Fig. 4. Mix-net based liquid democracy scheme $\Pi_{\textsc{Mix-Liquid}}$ in $\{\bar{\mathcal{G}}_{\mathrm{BB}}, \widehat{\mathcal{F}}_{\textsc{Cert}}\}$-hybrid world (Part II)

Ballot Casting Phase. As depicted in Fig. 4, the ballot casting phase consists of two rounds. In the first round, each voter V_i, $i \in [n]$ first fetches the trustees' partial public keys $\{\overline{\mathsf{pk}}_j\}_{j=1}^{k}$ 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 $\mathsf{pk} \leftarrow \mathsf{TRE.CombinePK}(\{\overline{\mathsf{pk}}_j\}_{j=1}^{k})$. In addition, the voter V_i picks a random temporary ID $w_i \leftarrow \{0,1\}^{\lambda}$. She then uses the

Tally (Part I)

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

∘ Round 1 to k:

- If $j = 1$, send $(\text{READ}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\text{BB}}$, and obtain $(\text{READ}, \mathsf{sid}, \mathit{state})$ 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 $(\text{VERIFY}, \mathsf{sid}, \mathsf{ssid}, (W_\ell, \pi_\ell^{(2)}), \sigma_\ell^{(2)})$ to $\widehat{\mathcal{F}}_{\text{CERT}}$, and receive $(\text{VERIFIED}, \mathsf{sid}, \mathsf{ssid}, (W_\ell, \pi_\ell^{(2)}), b_j^{(2)})$ from $\widehat{\mathcal{F}}_{\text{CERT}}$;
  - If $\langle \mathsf{ssid}, (U_\ell, V_\ell, \pi_\ell^{(3)}, \pi_\ell^{(4)}), \sigma_\ell^{(3)} \rangle$, is contained in state, then send $(\text{VERIFY}, \mathsf{sid}, \mathsf{ssid}, (U_\ell, V_\ell, \pi_\ell^{(3)}, \pi_\ell^{(4)}), \sigma_\ell^{(3)})$ to $\widehat{\mathcal{F}}_{\text{CERT}}$, receive $(\text{VERIFIED}, \mathsf{sid}, \mathsf{ssid}, (U_\ell, V_\ell, \pi_\ell^{(3)}, \pi_\ell^{(4)}), b_j^{(3)})$ from $\widehat{\mathcal{F}}_{\text{CERT}}$;

  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 $b_\ell^{(2)} \cdot b_\ell^{(3)} = 1$;
  - $\mathsf{NIZK}_{\mathcal{R}_5}.\mathsf{Verify}((\mathsf{pk}, W_\ell), \pi_\ell^{(2)}) = 1$;
  - $\mathsf{NIZK}_{\mathcal{R}_6}.\mathsf{Verify}((\mathsf{pk}, (W_0, \ldots, W_n), V_\ell), \pi_\ell^{(3)}) = 1$;
  - $\mathsf{NIZK}_{\mathcal{R}_5}.\mathsf{Verify}((\mathsf{pk}, U_\ell), \pi_\ell^{(4)}) = 1$;

  (Set $n' := i$ after the above process.)
- (If $j > 1$, T_j sends $(\text{READ}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\text{BB}}$, and obtain $(\text{READ}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\text{BB}}$; T_j then fetches $(e_{i,t}^{(j-2)})_{i=1}^{n'}, e_{i,t}^{(j-1)})_{i=1}^{n'}, \pi_{j-1}^{(5)}$ from state and check $\mathsf{NIZK}_{\mathcal{R}_7}.\mathsf{Verify}((\mathsf{pk}, (e_{1,t}^{(j-1)}, \ldots, e_{n',t}^{(j-1)}), (e_{1,t}^{(j)}, \ldots, e_{n',t}^{(j)})), \pi_j^{(5)}) = 1$, for $t \in [3]$.)
  T_j randomly picks a permutation Π_j over [n']; For $i \in [n']$, set $e_{i,1}^{(j)} \leftarrow \mathsf{TRE.ReRand}(\mathsf{pk}, e_{\Pi_j(i),1}^{(j-1)}; r_{i,1}^{(j)})$, $e_{i,2}^{(j)} \leftarrow \mathsf{TRE.ReRand}(\mathsf{pk}, e_{\Pi_j(i),2}^{(j-1)}; r_{i,2}^{(j)})$, and $e_{i,3}^{(j)} \leftarrow \mathsf{TRE.ReRand}(\mathsf{pk}, e_{\Pi_j(i),3}^{(j-1)}; r_{i,3}^{(j)})$, where $r_{i,1}^{(j)}, r_{i,2}^{(j)}, r_{i,3}^{(j)}$ are fresh randomness. Compute

$$\pi_j^{(5)} \leftarrow \mathsf{NIZK}_{\mathcal{R}_7}\left\{ \begin{array}{r} \left(\mathsf{pk}, (e_1^{(j-1)}, \ldots, e_{n'}^{(j-1)}), (e_1^{(j)}, \ldots, e_{n'}^{(j)})\right), \\ \left(\Pi_j, (r_{i,1}^{(j)}, r_{i,2}^{(j)}, r_{i,3}^{(j)})_{i\in[n']}\right) \quad : \\ \forall i \in [n'] \ : \quad e_{i,1}^{(j)} = \mathsf{TRE.ReRand}\left(\mathsf{pk}, e_{\Pi_j(i),1}^{(j-1)}; r_{i,1}^{(j)}\right) \\ \bigwedge \ e_{i,2}^{(j)} = \mathsf{TRE.ReRand}\left(\mathsf{pk}, e_{\Pi_j(i),2}^{(j-1)}; r_{i,2}^{(j)}\right) \\ \bigwedge \ e_{i,3}^{(j)} = \mathsf{TRE.ReRand}\left(\mathsf{pk}, e_{\Pi_j(i),3}^{(j-1)}; r_{i,3}^{(j)}\right) \end{array} \right\}$$

- Send $(\text{SIGN}, \mathsf{sid}, \mathsf{ssid}, (e_{i,1}^{(j)}, e_{i,2}^{(j)}, e_{i,3}^{(j)})_{i=1}^{n'}, \pi_j^{(5)}))$ to $\widehat{\mathcal{F}}_{\text{CERT}}$ and receive $(\text{SIGNATURE}, \mathsf{sid}, \mathsf{ssid}, (e_{i,1}^{(j)}, e_{i,2}^{(j)}, e_{i,3}^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), \sigma_j^{(4)})$ from $\widehat{\mathcal{F}}_{\text{CERT}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\text{SUBMIT}, \mathsf{sid}, \langle \mathsf{ssid}, (e_{i,1}^{(j)}, e_{i,2}^{(j)}, e_{i,3}^{(j)})_{i=1}^{n'}, \pi_j^{(5)}, \sigma_j^{(4)} \rangle)$ to $\bar{\mathcal{G}}_{\text{BB}}$.

Fig. 5. Mix-net based liquid democracy scheme $\Pi_{\text{MIX-LIQUID}}$ in $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}\}$-hybrid world (Part III)

election public key pk to encrypt w_i as $W_i \leftarrow \mathsf{TRE.Enc}(\mathsf{pk}, w_i; \beta_i)$ with fresh randomness β_i. She also computes the corresponding NIZK

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

Tally (Part II)

∘ Round $k+1$:

- Send $(\text{READ}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\text{BB}}$, and obtain $(\text{READ}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\text{BB}}$. For $j \in [k]$, if $\langle \mathsf{ssid}, (e_{i,1}^{(j)}, e_{i,2}^{(j)}, e_{i,3}^{(j)})_{i=1}^{n'}, \pi_j^{(5)}, \sigma_j^{(4)} \rangle$ is contained in state, then send $(\text{VERIFY}, \mathsf{sid}, \mathsf{ssid}, (e_{i,1}^{(j)}, e_{i,2}^{(j)}, e_{i,3}^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), \sigma_\ell^{(4)})$ to $\widehat{\mathcal{F}}_{\text{CERT}}$, and receive $(\text{VERIFIED}, \mathsf{sid}, \mathsf{ssid}, (e_{i,1}^{(j)}, e_{i,2}^{(j)}, e_{i,3}^{(j)})_{i=1}^{n'}, \pi_j^{(5)}), b_j^{(4)})$ from $\widehat{\mathcal{F}}_{\text{CERT}}$; if $b_j^{(4)} = 1$, check $\mathsf{NIZK}_{\mathcal{R}_7}.\mathsf{Verify}((\mathsf{pk}, (e_{1,t}^{(j-1)}, \ldots, e_{n',t}^{(j-1)}), (e_{1,t}^{(j)}, \ldots, e_{n',t}^{(j)})), \pi_j^{(5)}) = 1$, for $t \in [3]$. If any of the above checks is invalid, halt.
- For $i \in [n']$, $t \in [3]$ compute $\overline{m}_{i,t}^{(j)} \leftarrow \mathsf{TRE.ShareDec}(\mathsf{pk}, \overline{\mathsf{sk}}_j, e_{i,t}^{(k)})$. and

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

- Send $(\text{SIGN}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,t}^{(j)}, \pi_{j,i,t}^{(6)})_{i \in [n'], t \in [3]})$ to $\widehat{\mathcal{F}}_{\text{CERT}}$ and receives $(\text{SIGNATURE}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,t}^{(j)}, \pi_{j,i,t}^{(6)})_{i \in [n'], t \in [3]}, \sigma_j^{(5)})$ from $\widehat{\mathcal{F}}_{\text{CERT}}$, where $\mathsf{ssid} = (\mathsf{T}_j, \mathsf{ssid}')$ for some ssid'.
- Send $(\text{SUBMIT}, \mathsf{sid}, \langle \mathsf{ssid}, (\overline{m}_{i,t}^{(j)}, \pi_{j,i,t}^{(6)})_{i \in [n'], t \in [3]}, \sigma_j^{(5)} \rangle)$ to $\bar{\mathcal{G}}_{\text{BB}}$.

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

- Send $(\text{READ}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\text{BB}}$, and and obtain $(\text{READ}, \mathsf{sid}, \mathit{state})$ from $\bar{\mathcal{G}}_{\text{BB}}$. For $j \in [k]$, if $\langle \mathsf{ssid}, (\overline{m}_{i,t}^{(j)}, \pi_{j,i,t}^{(6)})_{i \in [n'], t \in [3]}, \sigma_j^{(5)} \rangle$ is contained in state, send $(\text{VERIFY}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,t}^{(j)}, \pi_{j,i,t}^{(6)})_{i \in [n'], t \in [3]}, \sigma_j^{(5)})$ to $\widehat{\mathcal{F}}_{\text{CERT}}$, and receive $(\text{VERIFIED}, \mathsf{sid}, \mathsf{ssid}, (\overline{m}_{i,t}^{(j)}, \pi_{j,i,t}^{(6)})_{i \in [n'], t \in [3]}, b_j^{(5)})$ from $\widehat{\mathcal{F}}_{\text{CERT}}$. If $\prod_{j=1}^{k} b_j^{(5)} = 1$, for all $j \in [k], i \in [n'], t \in [3]$, check $\mathsf{NIZK}_{\mathcal{R}_8}.\mathsf{Verify}((e_{i,t}^{(k)}, \overline{m}_{i,t}^{(j)}, \overline{\mathsf{pk}}_i), \pi_{i,j,t}^{(6)}) = 1$. If any of the above checks is invalid, return $(\text{ERROR}, \mathsf{sid})$ to the environment $\mathcal{Z}$ and halt.
- For $i \in [n']$: compute $m_{i,t} \leftarrow \mathsf{TRE.ShareCombine}((k,k), e_{i,t}^{(k)}, \{\overline{m}_{i,t}^{(j)}\}_{j=1}^{k})$, $t \in [3]$; define $B_i := (m_{i,1}, m_{i,2}, m_{i,3})$.
- Calculate election result $\mathit{result} \leftarrow \mathsf{TallyProcess}(\{B_i\}_{i \in [n']})$, and return $(\text{READRESULTRETURN}, \mathsf{sid}, \mathit{result})$ to $\mathcal{Z}$.

Fig. 6. Mix-net based liquid democracy scheme $\Pi_{\text{MIX-LIQUID}}$ in $\{\bar{\mathcal{G}}_{\text{BB}}, \widehat{\mathcal{F}}_{\text{CERT}}\}$-hybrid world (Part IV)

to show she is the creator of this ciphertext. Voter V_i then signs and posts $(W_i, \pi_i^{(2)})$ to $\bar{\mathcal{G}}_{\text{BB}}$. In the second round, each voter V_i, $i \in [n]$ first fetches all the posted encrypted temporary IDs from $\bar{\mathcal{G}}_{\text{BB}}$, and checks their attached NIZK proofs. For any missing or invalid (encrypted) temporary IDs, the voters replace them with $\mathsf{TRE.Enc}(\mathsf{pk}, \perp; 0)$, which is the encryption of $\perp$ with trivial randomness. Moreover, the voters also defines $W_0 \leftarrow \mathsf{TRE.Enc}(\mathsf{pk}, \perp; 0)$. The statement for liquid democracy, s_i, can be parsed as either (i) $(\mathsf{V}_j, \perp)$ or (ii) $(\perp, v_i)$.

In Case (i) $(\mathsf{V}_j, \perp)$, i.e. delegating to voter V_j, the voter produces V_i as a re-randomized W_j and U_i as encryption of $\perp$. She then 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 $\bar{\mathcal{G}}_{\text{BB}}$.

In Case (ii) $(\perp, v_i)$, i.e. voting directly v_i, analogous to Case (ii), the voter produces 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 $\bar{\mathcal{G}}_{\mathrm{BB}}$.

Tally Phase. The tally phase is depicted in Figs. 5 and 6. The trustees first fetches (W_i, V_i, U_i) (which is viewed as the submitted ballot for voter V_i) from $\bar{\mathcal{G}}_{\mathrm{BB}}$ 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 $\mathsf{TRE.ShareDec}$. More specifically, each trustee will sign and post its decryption shares to $\bar{\mathcal{G}}_{\mathrm{BB}}$.

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

Remark 1. The re-randmonizable encryption (TRE) scheme used in this protocol can be replaced by a re-randomizable RCCA encryption scheme. Here RCCA is the short name for *replayable CCA* defined by Canetti, Krawczyk, and Nielsen [9]. Several RCCA constructions can be found in literature [11,12,19,28]. In our construction, it is possible to distribute a publicly verifiable RCCA encryption scheme, e.g. [12] and then use it as an enhanced version of TRE. Subsequently, $\mathsf{NIZK}_{\mathcal{R}_6}$ can be removed. Since the running time of proving/verifying $\mathsf{NIZK}_{\mathcal{R}_6}$ is linear in the number of voters n, it is more efficient to use RCCA instead of TRE for large n in practice.

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

4 Further Discussions

Statement Policy. We initiate the study of statement voting and liquid democracy in this work. Our statement voting concept can be significantly extended to support much richer ballot statements, which opens a door for designing a new class of e-voting schemes. A natural question to ask is what type of statements are allowed. For correctness, the (deterministic) $\mathsf{TallyProcess}$ function should be a symmetric function in the sense that its output does not depend on the order of the ballots to be counted. Moreover, the voting statement has a maximum running time restriction to prevent DoS, and it should not depend on partial tally

result. This is known as fairness. Namely, the statement execution cannot be conditional on the partial tally result at the moment when the ballot is counted. On the other hand, the statement can take input as external information oracles, such as News, Stock market, etc. When statement voting is integrated with a blockchain infrastructure, our scheme can be used to enable *offline voting* or *smart voting*. In particular, the voters may submit their statement ballot any time before the election on the blockchain; during the tally phase, the voter's ballots will be decrypted, and their statements will define their final votes based on the latest information provided by News oracles on the blockchain.

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

Trusted Setup. Typically, trusted setup assumptions[3] are required for constructing UC-secure e-voting systems. Common Reference String (CRS) and Random Oracle (RO) are two popular choices in practice. If an e-voting system uses CRS, then we need to trust the party who generates the CRS, which, in our opinion, is a stronger assumption than believing no adversary can break a secure hash function, e.g., SHA3. Therefore, in this work, we realize our liquid democracy voting system in the RO model. As a future direction, we will construct more solutions to liquid democracy. For example, an alternative approach is as follows: we first use 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; e.g., at least one honest MPC player.

Privacy and Coercion Resilience. Both statement voting and liquid democracy voting extend (deviate) from the conventional e-voting; therefore, the conventional privacy definitions are no longer suitable for these new types of voting schemes. For instance, if delegation loop is not allowed in the liquid democracy, how much voter privacy can be possibly achieved? We will investigate the privacy of statement voting and liquid democracy in depth in our future work.

Finally, we note that coercion resilience is critical in many scenarios. We will investigate this strong security requirement in our future work, too. Recently, Daian *et al.* [14] discussed the difficulty to achieve coercion resilience in the on-chain voting. We remark that Daian *et al.* only excluded a special class of voting protocols that "users can generate their own keys outside of a trusted environment". A potential approach is to follow our preliminary result [3]; there, very different technique has been explored for achieving coercion resilience: voters' keys and correlated secret information are generated *inside* a trusted hardware which cannot be obtained by the coercer.

Voter's Complexity. In our FHE-based and MPC-based solutions, the voter's complexity is constant in the number of ballots; the voting tally members have

[3] Most non-trivial functionalities (including the e-voting functionality) cannot be UC-realized in the plain model [6,8,10].

linear (or superlinear) complexity with respect to the number of voters, which is asymptotically the same as many existing voting schemes. In our mix-net based protocol, the voter's complexity is linear in the number of ballots; we remark that, this is our implementation choice for small scale, statement voting. As already discussed in Remark 1 in previous section, we can replace the TRE encryption with an RCCA encryption [11,12,19,28] to achieve better (i.e., constant) voter's complexity in the mix-net based protocol.

Acknowledgement. We thank Jeremy Clark and the anonymous reviewers for their constructive comments. The first author was partially supported by EPSRC grant EP/P034578/1. The second author was partially supported by NSF award #1801470. This work is also supported by Ergo platform, Fractal Platform, and Blockchain institute.

A Security Definition for TRE

Definition 3. *We say* $\mathsf{TRE} = \{\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CombinePK}, \mathsf{CombineSK}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{ReRand}\}$ *is a* secure *threshold re-randomizable public key encryption if the following properties hold:*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

References

1. Adhocracy. Adhocracy official website. Accessed 21 Oct 2017
2. Adida, B.: Helios: web-based open-audit voting. In: USENIX Security (2008)
3. Alwen, J., Ostrovsky, R., Zhou, H.-S., Zikas, V.: Incoercible multi-party computation and universally composable receipt-free voting. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 763–780. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_37
4. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 175–196. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10879-7_11
5. Ben-Sasson, E., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 459–474. IEEE Computer Society Press, May 2014
6. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
7. Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003). http://eprint.iacr.org/2003/239
8. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_2

9. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_33
10. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 68–86. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_5
11. Chaidos, P., Cortier, V., Fuchsbauer, G., Galindo, D.: Beleniosrf: a non-interactive receipt-free electronic voting scheme. In: CCS 2016, pp. 1614–1625. ACM, New York (2016)
12. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable proof systems and applications. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 281–300. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_18
13. Chaum, D., Ryan, P.Y.A., Schneider, S.: A practical voter-verifiable election scheme. In: di Vimercati, S.C., Syverson, P., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 118–139. Springer, Heidelberg (2005). https://doi.org/10.1007/11555827_8
14. Daian, P., Kell, T., Miers, I., Juels, A.: On-Chain Vote Buying and the Rise of Dark DAOs (2018). http://hackingdistributed.com/2018/07/02/on-chain-vote-buying/
15. Degrave, J.: Getopinionated. GitHub repository. Accessed 21 Oct 2017
16. Ford, B.: Delegative democracy (2002). http://www.brynosaurus.com/deleg/deleg.pdf
17. Froelicher, D., et al.: Unlynx: a decentralized system for privacy-conscious data sharing. Proc. Privacy Enhancing Technol. **4**, 152–170 (2017)
18. Groth, J.: Evaluating security of voting schemes in the universal composability framework. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 46–60. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24852-1_4
19. Groth, J.: Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 152–170. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_9
20. Hardt, S., Lopes, L.: Google votes: a liquid democracy experiment on a corporate social network. Technical Disclosure Commons (2015). http://www.tdcommons.org/dpubs_series/79
21. Kiayias, A., Zacharias, T., Zhang, B.: DEMOS-2: scalable E2E verifiable elections without random oracles. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 352–363. ACM Press, October 2015
22. Kiayias, A., Zacharias, T., Zhang, B.: End-to-end verifiable elections in the standard model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 468–498. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_16
23. Kulyk, O., Marky, K., Neumann, S., Volkamer, M.: Introducing proxy voting to helios. In: ARES, pp. 98–106. IEEE Computer Society (2016)
24. Kulyk, O., Neumann, S., Marky, K., Budurushi, J., Volkamer, M.: Coercion-resistant proxy voting. In: ICT Systems Security and Privacy Protection (2016)
25. Kulyk, O., Neumann, S., Marky, K., Volkamer, M.: Enabling vote delegation for boardroom voting. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 419–433. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_26
26. LiquidFeedback. LiquidFeedback official website. Accessed 21 Oct 2017

27. Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 373–392. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_22
28. Prabhakaran, M., Rosulek, M.: Rerandomizable RCCA encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 517–534. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_29
29. Democracy Earth. The social smart contract. An open source white paper, 1 September 2017. Accessed 21 Oct 2017
30. Unruh, D., Müller-Quade, J.: Universally composable incoercibility. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 411–428. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_22
31. Zhai, E., Wolinsky, D.I., Chen, R., Syta, E., Teng, C., Ford, B.: Anonrep: towards tracking-resistant anonymous reputation. In: NSDI 2016, pp. 583–596 (2016)