Exploring boundaries of privacy in e2e verifiable e-voting systems.

Tamara Finogina

1 Introduction

In an electronic voting (e-voting) execution, the voters engage in an interaction with the system by providing sensitive data such as their vote preference, authentication passwords, or personal data used for election auditing. All collected data should be processed in a way that election integrity and voter privacy are preserved at the best possible level. In scenarios that e-voting runs at a national level data processing scales at the order of millions, whereas any security breach may lead to a massive and disastrous effect. Consequently, formal analysis and provable security of e-voting systems has been in the centre of related literature.

Voter privacy suggests that voters are capable of casting their votes secretly and freely without letting adversarial parties to learn any information about their preferences. On the other hand, integrity is traditionally captured by the end-to-end (E2E) verifiability notion states that the voter can obtain a receipt at the end of the ballot casting procedure that is used for verifying that his vote was (1) cast as intended, (2) recorded as cast, and (3) tallied as recorded. Furthermore, anyone should be able to verify that the election procedure is executed properly. It has been observed that voter privacy and E2E verifiability requirements inherently contradict each other at some point. Therefore, there should exist the maximum level of privacy that is possible to achieve in any E2E verifiable e-voting system.

2 Preliminaries

2.1 Notations

Through this paper, we denote λ as the security parameter. We use $negl(\lambda)$ to denote a negligible function in λ, i.e., it always holds that $negl(\lambda) < \frac{1}{\lambda^c}$ for any $0 < c \in \mathbb{Z}$ for sufficient large λ.

3 E-voting model

3.1 Syntax

An e-voting system is a tuple of interactive protocols and algorithms Π among a set of entities and voters, that takes voters' preferences as and input and aims to return a tally.

In this modelling, the election system involves five types of entities, the voters $V_1, \ldots, V_n$, possibly equipped with the voting supporting device (VSD) and the auditing supporting device (ASD), the election authority (EA), the vote collector (VC), the trustee (T), and the bulletin board (BB) whose only role is to provide storage for the election transcript for the purpose of verification.

(1) BB is completely passive and is only writeable by the VC, T and EA and readable by anyone.

(2) Voters submit their votes by engaging in the ballot casting protocol to the VC and they are not allowed to interact with each other.

(3) VC only role is to collect votes and write them to BB

(4) T is responsible for computing the tally and announcing the election result.

(5) EA - prepares all the election setup information and distributes the voters' ballots.

Protocols An e-voting system Π is a quintuple of algorithms and protocols **(Setup, Cast, Tally, Result, Verify)** specified as follows:

(1) The interactive protocol **Setup** is executed by the EA and T . During the setup phase EA generates Π's public parameters Pub (which include

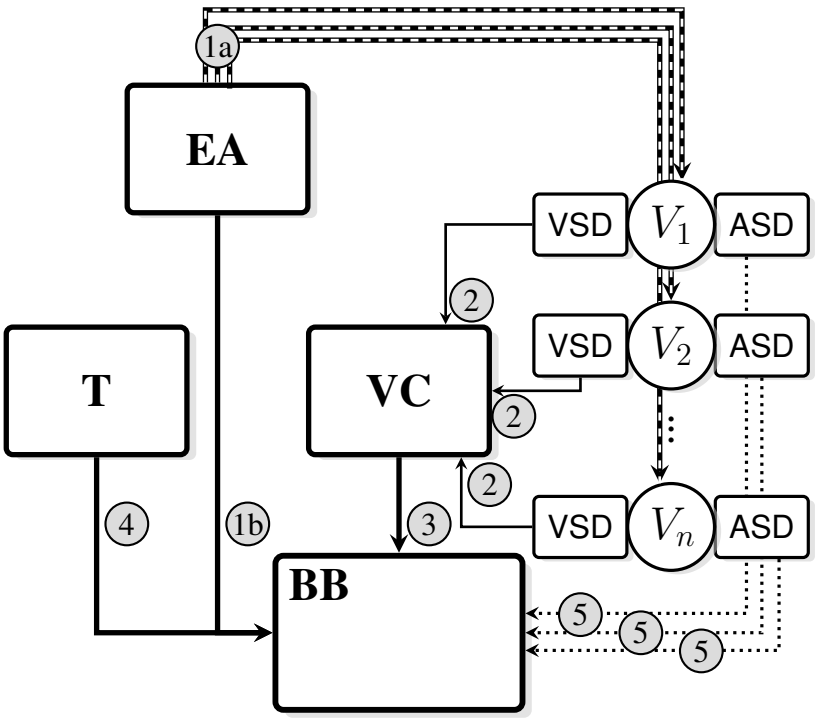


FIGURE 1: The interaction among the entities in an e-voting execution. The dotted lines denote read-only access to the BB. The dashed arrows denote channels for voters' private inputs distribution. Annotation: (1a): distribution of voter's private inputs; (1b): pre-election BB data; (2): vote casting; (3): writing votes to BB; (4) post-election BB data; (5): auditing.

P, V, U) and the voters' secrets $s_1, \ldots, s_n$. The part of the interactive protocol during which EA distributes secrets among voters is defined as **Registration**. At the same time, T generates pre-election BB data and posts it on BB.

(2) The interactive protocol **Cast** is executed between three parties, the voter V_l, the BB and the VC. During this interaction, the voter uses VSD, his secret s_l and an option U_l to generate the ballot b_l and sends this ballot to VC. Upon successful termination, VC posts ballot b_l to BB and the voter V_l receives a receipt α_l.

(3) The algorithm **Result** is executed by T and outputs the result τ for the election or returns $\perp$ in case such result is undefined.

(4) The algorithm **Verify** on input α, τ outputs a value in $\{0, 1\}$, where α is a voter receipt (that corresponds to the voter's output from the **Cast** protocol).

In some e-voting systems **Registration** part is omitted and voters are expected to receive their credentials via secure channel such as post mail, pulling place etc.

3.2 Correctness of a system

We say that a system Π has (perfect) correctness, if for any honest execution of any subset of not abstained voters that results in a public transcript τ, where the voters $V_1, ..., V_n$ cast votes for options $U_1, ..., U_n$, it holds that $Result(\tau) = f(U_1, ..., U_n)$, where $f(U_1, ..., U_n)$ is the m-vector whose i-th location is equal to the number of times a candidate $P_i \in \{P_1, \ldots, P_m\}$ was chosen in the candidate selections $U_1, ..., U_n$.

4 Privacy

During the election process an voter uses his credentials to cast his option by running the **Cast** protocol on a VSD. Since voters are not allowed to have or share a secret that would have helped them to preserve their privacy in case of a corrupted e-voting system, their privacy relies on the trusted entities of the system. There are two wildly known classes of e-voting systems: code-based system and encryption-based systems. The former relies on crypto that run by the trusted administrator in advance, the latter places its safety on crypto performed inside VSD

during the **Cast** protocol execution. From a voter point of view it means that his privacy is protected by trusted administrator and pre-calculated credentials, that would not reveal any sensitive information even if the choice is sent in a plane text, or by trusted VSD and crypto performed inside it.

The approach that we used in this thesis is the following: an honest voter is allowed to have **only one** perfectly hidden from an adversarial eyes interaction and at the end he provides the adversary with the real and simulated view of the result of this interaction. $\mathcal{A}$ is allowed to observe a network trace of all interactions and play on behalf of corrupted entities and voters. This one perfectly private interaction can be either an act of actually entering voter's preference into VSD or while a voter receives his credentials. If $\mathcal{A}$ has no advantage in distinguishing real and simulated view over a coin flip, the system is considered private.

We formally define the voter privacy via a Voter Privacy game, denotes as $G^{\mathcal{A},Sim}_{t-priv,<honest\ entities>}(1^\lambda)$, that is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, that takes as input the security parameter λ and returns 1 or 0 depending on whether the adversary wins. Also, $\mathcal{A}$ is allowed to corrupt some entities. The choice of the corrupted parties splits the Voter Privacy game into two different scenarios: (1) entities (EA, T) are hones and (2) VSD is honest.
The only difference between Strict privacy and Privacy is the number of corrupted voters. In Strict privacy games **all** voters are corrupted and $\mathcal{A}$ vote on their behalf. On the contrary, in the Privacy games at most t voters are corrupted and all others are honest. The crucial moment is that honest voters are able to verify that their vote was cast as intended, but an adversary can not since he does not know the intent and does not control all entities. Therefore, voters may lie about vote while an adversary have no ways to learn the truth and not to win the game $G^{\mathcal{A},Sim}_{t-priv,<honest\ entities>}(1^\lambda)$.

4.1 EA and T are honest: $G^{\mathcal{A},Sim}_{t-priv,\mathrm{EA},T}(1^\lambda)$

In the game $G^{\mathcal{A},Sim}_{t-priv,\mathrm{EA},T}(1^\lambda)$, an adversary $\mathcal{A}$ interacts with the challenger $\mathcal{C}$ on behalf of all corrupted voters, VC and VSD. $\mathcal{C}$ plays the role of honest voters, EA and T. BB is completely passive and represents a publicly viewed database.

After an adversary $\mathcal{A}$ defined an election parameters, $\mathcal{C}$ doubles the voters, computes setup data and starts the election. This duplication of voters is needed be-

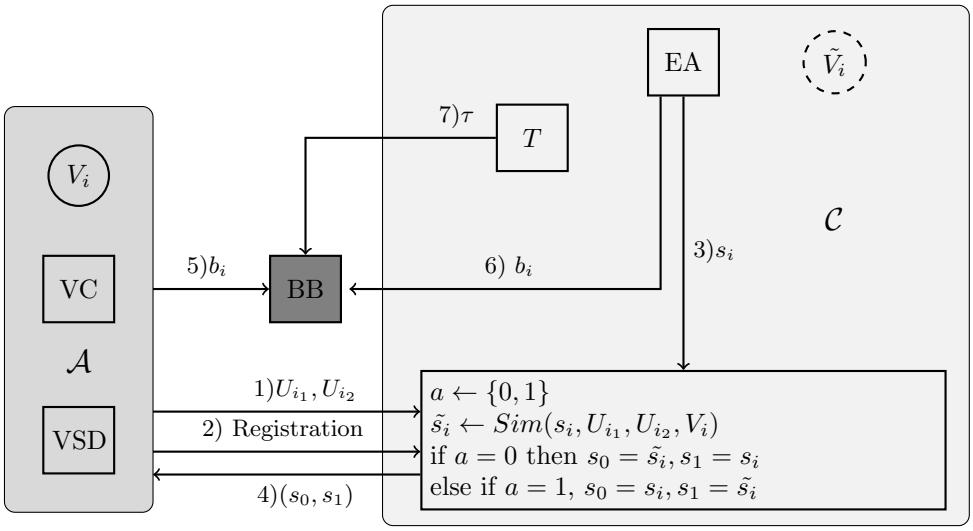


FIGURE 2: $G^{\mathcal{A},Sim}_{t-priv,\mathrm{EA},T}(1^\lambda)$

cause $\mathcal{C}$ would post another ballot for every adversarial's cast ballot. If $\mathcal{A}$ abstains, $\mathcal{C}$ abstains as well. During the game, we will refer to duplicated voters as fake ones.

$\mathcal{A}$ 1) picks and sends two options U_{i_1}, U_{i_2} to the challenger $\mathcal{C}$. One of the options U_{i_1}, U_{i_2} is his intent, the other – the option that the challenger would use in order to produce an indistinguishable from the intent's ballot and receipt view*. After sending options, 2) $\mathcal{A}$ schedules the **Registration** protocol with $\mathcal{C}$ on behalf of some voter V_i. $\mathcal{C}$ creates a fake credentials $\tilde{s_i}$ and 3) generates real credentials s_i for him. 4) $\mathcal{C}$ responds $\mathcal{A}$ with a pair of credentials s_0, s_1, where one of the credentials are real and the other were generated using the simulator Sim in a such way, that no matter what credential s_0 or s_1 $\mathcal{A}$ uses to cast a vote for U_{i_1}, the produced ballot would also correspond to the result of the **Cast** protocol for an option U_{i_2} with the other credentials and vice versa. If 5) $\mathcal{A}$ choses to post the ballot b_i to BB, 6) $\mathcal{C}$ posts exactly the same ballot to BB on behalf of the fake voter. When $\mathcal{A}$ stops the election, 7) $\mathcal{C}$ posts the tally τ **.

Remarks:
*If $\mathcal{C}$ succeeds, $\mathcal{A}$ wouldn't be able to say whether his ballot and receipt corresponds to U_{i_1} or U_{i_2} and BB would contain both ballots (one for the option U_{i_1}

,the other for the option U_{i_2}) so the tally wouldn't reveal any information. Example: if $\mathcal{A}$ picks real credential for the voter V_i then he casts a vote for the option U_{i_1} , at the same time exactly the same ballot posted $\mathcal{C}$ on behalf of the fake voter $\tilde{V}_i$ would correspond to the fake credentials and the option U_{i_2}. In this case $\mathcal{A}$ voted for U_{i_1}. Else if $\mathcal{A}$ picks fake credentials and tries to vote for the option U_{i_1}, his ballot corresponds to the real credentials and the option U_{i_1}. At the same time, exactly the same ballot posted $\mathcal{C}$ on behalf of the fake voter $\tilde{V}_i$ would correspond to the fake credentials and the option U_{i_1}. So $\mathcal{A}$ voted for U_{i_1}. If this definition holds, $\mathcal{A}$ has no idea whom he voted for.
** The tally τ is posted only if all ballots are correctly formed: were produced with one of the provided credentials, for the candidate U_{i_1}, for every cast ballot there is a duplicate, etc.

$G^{\mathcal{A},Sim}_{t-priv,\mathrm{EA},T}(1^\lambda)$ defined as follows:

(1) During the game $\mathcal{C}$ plays the role of honest voters and EA. $\mathcal{A}$ operates on behalf of corrupted voters and VSD.

(2) $\mathcal{A}$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$, a set of allowed candidates' selections $\mathcal{U}$. It provides $\mathcal{C}$ with $\mathcal{V}, \mathcal{P}, \mathcal{U}$.

(3) $\mathcal{C}$ doubles the set $\mathcal{V}$ adding fake voters $\{V'_1, ..., V'_n\}$ and starts the election on behalf of EA. Also, $\mathcal{C}$ flips a coin $a \leftarrow \{0, 1\}$ to define an order according to which real and simulated credentials would be returned to $\mathcal{A}$.

(4) The adversary $\mathcal{A}$ picks two option $U_{i_1}, U_{i_2} \in \mathcal{U}$, where U_{i_1} is an option for the real credentials and U_{i_2} is an option for the fake ones. After that, $\mathcal{A}$ and $\mathcal{C}$ engage in an interaction where $\mathcal{A}$ schedules the $Registration$ protocols, during which all voters receive their credentials and forward them to $\mathcal{A}$. For each voter $V_i \in \mathcal{V}$, the adversary chooses whether V_i is corrupted:

– If V_i is corrupted, then $\mathcal{C}$ provides $\mathcal{A}$ with the real credentials s_i, and then they engage in a $Cast$ protocol where A vote on behalf of V_i and $\mathcal{C}$ plays the role of EA.

– If V_i is not corrupted, then $\mathcal{C}$ generates real credentials s_i. Also $\mathcal{C}$ picks a fake voter $\tilde{V}_i$ and generates credentials $\tilde{s}_i$ for him using Sim*.

$\mathcal{C}$ responds $\mathcal{A}$ with a pair of simulated and real credentials (s_0, s_1) in order defined by the coin a: $\begin{cases} \text{if } a = 0, \ (s_0, s_1) = (\tilde{s_i}, s_i) \\ \text{else } \ (s_0, s_1) = (s_i, \tilde{s_i}) \end{cases}$

– Honest voters forward both credentials to $\mathcal{A}$

– Using one of the credentials $\mathcal{A}$ schedules the $Cast$ protocol executions and sends the produced ballot to BB.

– If $\mathcal{A}$ posts a ballot on BB, $\mathcal{C}$ posts exactly the same ballot in the entry that corresponds to the fake voter $\tilde{V_i}$

(5) $\mathcal{C}$ executes the $Tally$ protocol.

(6) Finally, $\mathcal{A}$ using all information collected above (including the contents of the BB) outputs a bit a^*

(7) Denote the set of corrupted voters as $\mathcal{V}_{corr}$ and the set of honest voters as $\tilde{\mathcal{V}} = \mathcal{V} \backslash \mathcal{V}_{corr}$. The game returns a bit which is 1 if and only if the following hold true:

(a) $a = a^*$

(b) $|\mathcal{V}_{corr}| \leq t$ (i.e., the number of corrupted voters is bounded by t).

***Remark:**
Sim works in a way that the fake credentials satisfy both of the following rules:

(1) The real credentials and U_{i_1} option should give ballot and receipt, which are identical* to ballot and receipt produced for the fake credentials and U_{i_2} option.

(2) The fake credentials and U_{i_1} option should give ballot and receipt, which are identical to ballot and receipt produced for the real credentials and U_{i_2} option.

Privacy: EA **and** T **are honest:**

The e-voting system Π achieves voter privacy in case of honest T and EA for at most t corrupted voters if there is a PPT simulator Sim such that for any PPT adversary $\mathcal{A}$:

$$|\Pr[G^{\mathcal{A},Sim}_{t-priv,\mathrm{EA},T}(1^\lambda) = 1] - \tfrac{1}{2}| = negl(\lambda)$$

4.2 VSD is honest: $G^{\mathcal{A},Sim}_{t-priv,\mathrm{VSD}}(1^\lambda)$

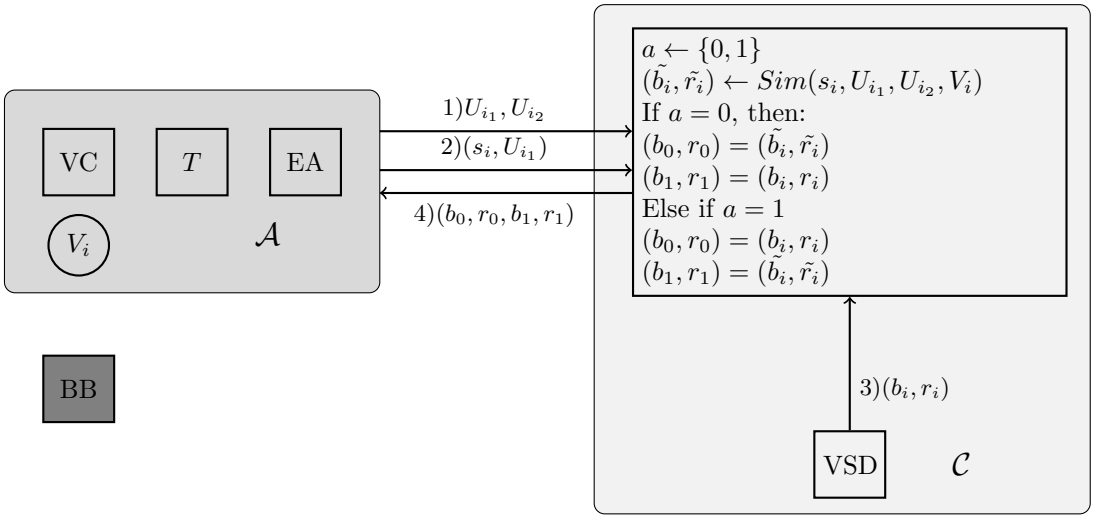


FIGURE 3: $G^{\mathcal{A},Sim}_{t-priv,\mathrm{VSD}}(1^\lambda)$

In the game $G^{\mathcal{A},Sim}_{t-priv,\mathrm{EA},T}(1^\lambda)$, an adversary $\mathcal{A}$ operates on behalf of the all voters and all corrupted entities, such as: EA, VC and T. BB is completely passive and represents a publicly accessible database.
1) an adversary $\mathcal{A}$ picks and sends options U_{i_1}, U_{i_2} to the challenger $\mathcal{C}$. After that 2) $\mathcal{A}$ schedules the **Cast** protocol with $\mathcal{C}$ on behalf of some voter V_i*. $\mathcal{C}$ 3) generates a real ballot and receipt and uses Sim to create a fake ones. At the end 4) $\mathcal{C}$ responses with a pair of ballots and receipts b_0, r_0, b_1, r_1, where one ballot and receipt corresponds to an option U_{i_1}, the other was generated using the simulator Sim for an option U_{i_2}.

*Remark:
Existence of honest voters is the only difference from the Strict privacy definition. In the strict privacy game, all voters are corrupted.

The game $G^{\mathcal{A},Sim}_{t-priv,\mathrm{VSD}}(1^\lambda)$ is defined as follows:

(1) During the game $\mathcal{C}$ plays the role of the honest voters and VSD. $\mathcal{A}$ operates on behalf of corrupted voters and may corrupt T, and EA.

(2) $\mathcal{A}$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$, a set of allowed candidates' selections $\mathcal{U}$ and starts the election.

(3) $\mathcal{C}$ flips a coin $a \leftarrow \{0, 1\}$ to define an order according to which real and simulated ballots and receipts would be returned to $\mathcal{A}$.

(4) $\mathcal{A}$ sends to $\mathcal{C}$ options $U_{i_1}, U_{i_2} \in \mathcal{U}$, where U_{i_1} is an option for the real ballot and receipt and U_{i_2} is an option for the fake ones. After that, $\mathcal{A}$ and $\mathcal{C}$ engage in an interaction where $\mathcal{A}$ schedules the **Cast** protocols of all voters which may run concurrently. For each voter $V_i \in \mathcal{V}$, the adversary chooses whether $V_i \in \mathcal{V}$ is corrupted:

– If V_i is corrupted, then $\mathcal{C}$ provides $\mathcal{A}$ with the real ballot and receipt (b_i, r_i).

– If V_i is not corrupted, $\mathcal{C}$ provides $\mathcal{A}$ with provides $\mathcal{A}$ with a pair of simulated and real ballot and receipt $(b_0, r_0)(b_1, r_1)$ s.t.:
$$\begin{cases} \text{if } \ a = 0, \ \ (b_0, r_0) = (\tilde{b_i}, \tilde{r_i}) \ \text{ and } \ (b_1, r_1) = (b_i, r_i) \\ \text{else } \ (b_0, r_0) = (b_i, r_i) \ \text{ and } \ (b_1, r_1) = (\tilde{b_i}, \tilde{r_i}) \end{cases}$$
where the pair (b_i, r_i) is the ballot and receipt for an adversarial option U_{i_1} and $(\tilde{b_i}, \tilde{r_i})$ is the ballot and receipt for U_{i_2} option generated via the simulator Sim.

(5) Finally, (if T is corrupted) $\mathcal{A}$ executes the $Tally$ protocol and using all information collected above outputs a bit a^*

(6) Denote the set of corrupted voters as $\mathcal{V}_{corr}$ and the set of honest voters as $\tilde{\mathcal{V}} = \mathcal{V} \backslash \mathcal{V}_{corr}$. The game returns a bit which is 1 if and only if the following hold true:

(a) $a = a^*$

(b) $|\mathcal{V}_{corr}| \leq t$ (i.e., the number of corrupted voters is bounded by t).

Privacy: VSD **is honest:**

The e-voting system Π achieves voter privacy in case of honest VSD, for at most t corrupted voters , if there is a PPT simulator Sim such that for any PPT adversary

$\mathcal{A}$:

$|\Pr[G^{\mathcal{A},Sim}_{t-priv,\mathrm{VSD}}(1^\lambda) = 1] - \frac{1}{2}| = negl(\lambda)$

4.3 Comparison with existing definitions

4.3.1 Helious. Ballot privacy.

Ballot privacy attempts to capture the idea that during its execution a secure protocol does not reveal information about the votes cast, beyond what the result of the election leaks. In some works, ballot privacy defined even stronger: "a voter's vote is not revealed to anyone". However, in most cases ballot privacy targets specifically vote-casting procedure end

Informally, ballot privacy is satisfied if an adversary in control of arbitrarily many voters cannot distinguish between real ballots and fake ballots, where ballots are replaced by ballots for some fixed vote ϵ chosen by adversary. The adversary $\mathcal{A}$ has read access to public BB and may observe communication channels between the honest parties and BB. Note, that $\mathcal{A}$ is not allowed to corrupt any entities.

Definition: Ballot privacy for Helious:
The challenger $\mathcal{C}$ starts by flipping a coin a, which defines in what world the game between $\mathcal{C}$ and an adversary $\mathcal{A}$ would take place. If $a = 0$, the world is real, otherwise – fake. Also $\mathcal{C}$ maintains two bulletin boards $\mathrm{BB}, \mathrm{BB}'$ initialized via the setup algorithm, where BB' always contains ballots for the real votes. The adversary $\mathcal{A}$ is always given access BB and can issue two types of queries: **vote** and **ballot**. In the real world a **vote** query causes a ballot for the given vote to be placed on the both BB: hidden BB' and public BB. In the fake one, the same query causes a ballot for the given vote to be placed on the BB' and a ballot for ϵ to be placed on BB. A **ballot** query always causes the submitted ballot to be processed on both boards. At some point, the adversary $\mathcal{A}$ asks to see the result. The challenger computes tally based on BB'. The adversarial goal is to determine whether the world is real or fake.

The Ballot privacy for Helious contradicts verifiability notion by nature. Intuitively, verifiability means that it's possible to check that a vote was cast as intended, recorded as cast, tallied as recorded and, if the tally is encrypted, final

result was encrypted as required. The definition states that an adversary can not distinguish real and fake world, assuming that he observes communications channels and has access to the public BB only. In general case BB and BB' contains different sets of votes, though tallying is always done using BB'. The result of an election corresponds to evaluating an arbitrary function ρ that takes a list of votes as input and returns the election result on the underlying votes. Suppose, that there is a proof π that the result was tallied as recorded. π guarantees that tallying procedure was performed on the given BB and non vote has been modified or excluded. If there is such proof, an adversary agains ballot privacy could easily check that produced result, even if decrypted correctly, was computed for some other BB and therefore guess the challenger's coin a with an overwhelming probability.

To defend against this attack, $\mathcal{C}$ should be able to fake proof π. Suppose there is a simulator that can fake the proof π without using a global setup. That would mean that secure schemes would not satisfy tally uniqueness, since simulator allows any result to be accepted as the valid one. So , the same BB would have multiple valid election results, which contradicts verifiability.

Another case - using global setup or trapdoors.

4.3.2 Demos privacy

Demos voter privacy definition resembles witness indistinguishability of interactive proof system. An adversary's challenge is to distinguish between two predefined by the adversary $\mathcal{A}$ lists of candidate selection that sums up to the same tally.

The game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ are defined as follows: The adversary defines election parameters: voters, candidates and selects two lists of candidates' selections $\mathcal{L} =< U_l^0, U_l^1 >$ that sums up to the same tally. The challenger flips a coin b and starts an election. During the game, $\mathcal{A}$ schedules all **Cast** protocols selecting corrupted voters adaptively ($\mathcal{A}$ allowed to corrupt t voters at most). For all honest voters, $\mathcal{A}$ provides $\mathcal{C}$ with two candidates selections U_l^0, U_l^1. $\mathcal{C}$ selects U_l^b as voting option, runs the **Cast** protocol and returns to the adversary (i) the receipt r_l obtained from the protocol, and (ii) if $b = 0$ current view obtained from the protocol or if $b = 1$, a simulated view produced by a simulator $\mathcal{S}$.

According to the game, for a voter V_l, if $b = 0$ $\mathcal{A}$ receives back a receipt r_l the first candidates' selection U_l^0 and the current real view of the internal state of the voter obtained from the **Cast** protocol. Otherwise, for $b = 1$, $\mathcal{A}$ gets back a receipt r_l the second candidates' selection U_l^1 and the simulated view generated by a simulator $\mathcal{S}$. The e-voting scheme Π with at most t corrupted voters achieves voter privacy if there exist a simulator $\mathcal{S}$ such that $\mathcal{A}$ has a negligible advantage over a random coin flipping in guessing b.

Demos privacy definition
$G^{*\mathcal{A},\mathcal{S}}_{DEMOS,t-priv}(1^\lambda, n, m)$ defined as follows:

During the game $\mathcal{C}$ plays the role of honest voters, T and EA. $\mathcal{A}$ operates on behalf of corrupted voters and VSD.

(1) $\mathcal{A}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, choses a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$ and the set of allowed candidates' selections $\mathcal{U}$. It provides $\mathcal{C}$ with $\mathcal{V}, \mathcal{P}, \mathcal{U}$.

(2) $\mathcal{C}$ flips a coin $b \in \{0, 1\}$ and perform the **Setup** protocol on input $1^\lambda, \mathcal{V}, \mathcal{P}, \mathcal{U}$ to obtain $msk, s_1, ..., s_n, Pub$; it provides $\mathcal{A}$ with Pub.

(3) The adversary $\mathcal{A}$ and $\mathcal{C}$ engage in an interaction where $\mathcal{A}$ schedules the **Cast** protocols of all voters which may run concurrently. For each voter $V_i \in \mathcal{V}$ the adversary chooses whether V_i is corrupted:

> – If V_i is corrupted, then $\mathcal{C}$ provides s_l to $\mathcal{A}$, and then they engage in a **Cast** protocol where $\mathcal{A}$ plays the role of V_i and $\mathcal{C}$ plays the role of EA and BB.
>
> – If V_i is not corrupted, $\mathcal{A}$ provides two candidates selections $\langle \mathcal{U}_l^0, \mathcal{U}_l^1 \rangle$ to the challenger $\mathcal{C}$. $\mathcal{C}$ operates on V_l's behalf, using $\mathcal{U}_l^b$ as the V_l's input. The adversary $\mathcal{A}$ is allowed to observe the network trace of the **Cast** protocol where $\mathcal{C}$ plays the role of V_l, EA and BB. When the **Cast** protocol terminates, the challenger $\mathcal{C}$ provides to $\mathcal{A}$: (i) the receipt α_l that voter V_l obtains from the protocol, and (ii) if b = 0, the current view of internal state of the voter V_l, $view_l$ that the challenger obtains from the **Cast** protocol execution; or a simulated view of the internal state of V_l produced by $\mathcal{S}(view_l)$.

(4) $\mathcal{C}$ performs the **Tally** protocol playing the role of EA, T and BB. $\mathcal{A}$ is allowed to observe the network trace of that protocol.

(5) Finally, $\mathcal{A}$ using all information collected above (including the contents of the BB) outputs a bit b^*

Denote the set of corrupted voters as $\mathcal{V}_{corr}$ and the set of honest voters as $\tilde{\mathcal{V}} = \mathcal{V} \backslash \mathcal{V}_{corr}$. The game returns a bit which is 1 if and only if the following hold true:

(1) $b = b^*$

(2) $|\mathcal{V}_{corr}| \leq t$ (i.e., the number of corrupted voters is bounded by t).

(3) $f(\langle \mathcal{U}_l^0 \rangle_{V_l \in \tilde{\mathcal{V}}}) = f(\langle \mathcal{U}_l^0 \rangle_{V_l \in \tilde{\mathcal{V}}})$ (i.e., the election result w.r.t. the set of voters $\tilde{\mathcal{V}}$ does not leak b).

To prove that Demos privacy implies a weaker level of privacy than we defined, we constructed an modified version of our privacy definition. The difference between the modified and original versions is that it is mandatory to use two lists that sums up to the same tally in the modified one.

Modified original privacy definition

The only difference between the original privacy definition and the modified version is that $\mathcal{A}$ picks its choices U_i^0 U_i^1 from the two lists that sums up to the same tally: $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}})$

$G^{\mathcal{A},Sim}_{mod\_ORIG,t-priv,\mathrm{EA},T}(1^\lambda, n, m)$ defined as follows:

(1) $\mathcal{A}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, choses a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$ and the set of allowed candidates' selections $\mathcal{U}$. It provides $\mathcal{C}$ with $\mathcal{V}, \mathcal{P}, \mathcal{U}$.

(2) $\mathcal{C}$ starts the election on behalf of EA. Also, $\mathcal{C}$ flips a coin $a \leftarrow \{0, 1\}$ to define an order according to which real and simulated credentials would be returned to $\mathcal{A}$.

(3) The adversary $\mathcal{A}$ picks two option $U_i^0, U_i^1 \in \mathcal{U}$, where U_i^0 is its intent and U_i^1 is an option that $\mathcal{C}$ would use in order to fool $\mathcal{A}$. After that, $\mathcal{A}$ and

$\mathcal{C}$ engage in an interaction where $\mathcal{A}$ schedules the **Registration** protocols, during which all voters receive their credentials and forward them to $\mathcal{A}$. For each voter $V_i \in \mathcal{V}$, the adversary chooses whether V_i is corrupted:

– If V_i is corrupted, then $\mathcal{C}$ provides $\mathcal{A}$ with the real credentials s_i, and then they engage in a **Cast** protocol where $\mathcal{A}$ vote on behalf of V_i and $\mathcal{C}$ plays the role of EA.

– If V_i is not corrupted, then $\mathcal{C}$ generates real credentials s_i. Also $\mathcal{C}$ generates fake credentials $\tilde{s_i}$ using Sim^*. $\mathcal{C}$ responds $\mathcal{A}$ with a pair of simulated and real credentials (s_0, s_1) in order defined by the coin a:
$$\begin{cases} \text{if } \ a = 0, \ \ (s_i^0, s_i^1) = (\tilde{s}_i, s_i) \\ \text{else } \ (s_i^0, s_i^1) = (s_i, \tilde{s}_i) \end{cases}$$

– An honest voter V_i forward both credentials s_i^0, s_i^1 to $\mathcal{A}$

– Using one of the credentials $\mathcal{A}$ schedules the **Cast** protocol execution to vote for an option U_i^0 and sends the produced ballot to BB in the entry that corresponds to the voter V_i. As a result of the **Cast** protocol execution, $\mathcal{A}$ would obtain: receipt r_i, ballot b_i and the view of the internal state of the voter V_i $view_i$. During the **Tally** protocol execution this ballot would be treated as if it was generated with real credentials s_i. So, if $\mathcal{A}$ indeed picked the real credentials, b_i would correspond to the option U_i^0 and $view_i$ would be real. Otherwise, b_i would be a ballot for the option U_i^1 and $view_i$ would be fake.

– If $\mathcal{A}$ posts a ballot on BB, $\mathcal{C}$ posts exactly the same ballot in the entry that corresponds to the fake voter $\tilde{V_i}$. During the **Tally** protocol execution this ballot would be treated as if it was generated with fake credentials, which means that whatever option in reality $\mathcal{A}$ voted for, $\mathcal{C}$ picked the other option.

(4) $\mathcal{C}$ executes the **Tally** protocol on the BB.

(5) Finally, $\mathcal{A}$ using all information collected above (including the contents of the BB) outputs a bit a^*

(6) Denote the set of corrupted voters as $\mathcal{V}_{corr}$ and the set of honest voters as $\tilde{\mathcal{V}} = \mathcal{V} \backslash \mathcal{V}_{corr}$. The game returns a bit which is 1 if and only if the following hold true:

   (a) $a = a^*$

(b) $|\mathcal{V}_{corr}| \leq t$ (i.e., the number of corrupted voters is bounded by t).

(c) $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}})$ (i.e., the election result w.r.t. the set of voters $\tilde{\mathcal{V}}$ does not leak a).

Suppose there exists an adversary $\mathcal{B}$ that wins the Demos privacy game with the probability more than one half. We will show that it's possible to construct an adversary $\mathcal{A}$ that exploits $\mathcal{B}$ and breaks the modified original privacy.

$\mathcal{B}$ – $\mathcal{A}$ – $\mathcal{C}$ (challenger against Modified original privacy) interaction:

(1) $\mathcal{B}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, choses a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$ and the set of allowed candidates' selections $\mathcal{U}$. It provides $\mathcal{A}$ with $\mathcal{V}, \mathcal{P}, \mathcal{U}$.

(2) $\mathcal{A}$ forwards to $\mathcal{C}$ lists $\mathcal{V}, \mathcal{P}, \mathcal{U}$ defined by $\mathcal{B}$

(3) $\mathcal{C}$ starts the election on behalf of EA. Also, $\mathcal{C}$ flips a coin $a \leftarrow \{0, 1\}$ to define an order according to which real and simulated credentials would be returned to $\mathcal{A}$.

(4) The adversary $\mathcal{A}$ and $\mathcal{C}$ engage in an interaction where $\mathcal{A}$ schedules the **Registration** protocols, during which all voters receive their credentials and forward them to $\mathcal{A}$. At the same time, $\mathcal{A}$ simulates the challenger for demos privacy game for $\mathcal{B}$ and schedules the **Cast** protocol with $\mathcal{B}$. For each voter $V_i \in \mathcal{V}$ the adversary $\mathcal{B}$ chooses whether V_i is corrupted and $\mathcal{A}$ forwards this decision to $\mathcal{C}$:

– If V_i is corrupted, then $\mathcal{C}$ provides s_i to $\mathcal{A}$, $\mathcal{A}$ forwards s_i to $\mathcal{B}$ and then they engage in a **Cast** protocol where $\mathcal{B}$ plays the role of V_i and $\mathcal{C}$ plays the role of EA and BB, while $\mathcal{A}$ simply transfers information from $\mathcal{B}$ to $\mathcal{C}$.

– If V_i is not corrupted, $\mathcal{B}$ provides two candidates selections $\langle \mathcal{U}_i^0, \mathcal{U}_i^1 \rangle$ to $\mathcal{A}$. $\mathcal{A}$ forwards this pair to $\mathcal{C}$. $\mathcal{C}$ generates real credentials s_i and fake credentials $\tilde{s_i}$ using Sim. $\mathcal{A}$ receives a pair of simulated and real credentials (s_0, s_1) in order defined by the coin a: $\begin{cases} \text{if } a = 0, \ (s_i^0, s_i^1) = (\tilde{s_i}, s_i) \\ \text{else } \ (s_i^0, s_i^1) = (s_i, \tilde{s_i}) \end{cases}$
$\mathcal{A}$ always uses the first credentials s_i^0 to schedule the **Cast** protocol execution and vote for an option U_i^0. Since $\mathcal{A}$ controls VSD, it produces

ballot b_i and posts it to the BB in the entry that corresponds to the voter V_i. As a result of the **Cast** protocol execution, $\mathcal{A}$ would obtain: receipt r_i, ballot b_i and the view of the internal state of the voter V_i $view_i$. During the **Tally** protocol execution this ballot would be treated as if it was generated with real credentials s_i. So, if $\mathcal{A}$ indeed picked the real credentials, b_i would correspond to the option U_i^0 and $view_i$ would be real. Otherwise, b_i would be a ballot for the option U_i^1 and $view_i$ would be fake. The adversary $\mathcal{B}$ is allowed to observe the network trace of the **Cast** protocol where $\mathcal{A}$ plays the role of V_l and VSD and $\mathcal{A}$ plays on behalf of EA and BB. When the **Cast** protocol terminates, $\mathcal{A}$ provides to $\mathcal{B}$: (i) the receipt r_i that voter V_i obtains from the protocol, and (ii) $view_i$

(5) $\mathcal{C}$ performs the **Tally** protocol playing the role of EA, T and BB. $\mathcal{A}$ and $\mathcal{B}$ are allowed to observe the network trace of that protocol.

(6) Finally, $\mathcal{B}$ using all information collected above (including the contents of the BB) outputs a bit b^*

(7) $\mathcal{A}$ returns $1 - b^*$

If the challenger's $\mathcal{C}$ coin $a = 0$, then the first credentials s_i^0 are always fake. Since $\mathcal{A}$ always uses s_i^0 to schedule the **Cast** protocol execution with $\mathcal{C}$ and vote for an option U_i^0, as a result of the execution $\mathcal{A}$ would obtain: receipt r_i, ballot b_i that is actually a vote for U_i^1 and the fake view of the internal state of the voter V_i $view_i$. This case corresponds to the coin $b = 1$ in the Demos privacy game.
Else if $\mathcal{C}$'s coin $a = 1$, s_i^0 are always real credentials. As a result of the **Cast** protocol execution $\mathcal{A}$ would obtain: receipt r_i, ballot b_i that is indeed a vote for U_i^0 and the real view of the internal state of the voter V_i $view_i$. This case corresponds to the coin $b = 0$ in the Demos privacy game.
By assumption $\mathcal{B}$ has an advantage over a coin flipping in winning the demos privacy game. This means that $\mathcal{B}$ is capable of guessing coin b with probability more than one half. $\mathcal{A}$ returns $1 - b^*$, where b^* is the $\mathcal{B}$'s guess. The above states that $\mathcal{A}$ is capable of wining the modified privacy game against the callenger $\mathcal{C}$ with probability more than random coin flipping, which means $\mathcal{A}$ breaks modified privacy definition.

So, if there is an adversary that breaks Demos privacy, there is an adversary that exploits it in order to break modified privacy.

5 Privacy with respect to entities

5.1 EA only

If EA is trusted and all other entities ($T, \mathrm{VSD}, \mathrm{VC}$) are corrupted ($\mathrm{BB}$ is completely passive and simply posts all information received from VC) it is impossible to preserve voter's privacy.

Proof:

Suppose there is some e-voting system Π which doesn't allow any adversary to learn anything about individual voters' intents in case when only EA is trusted. $\mathcal{A}$ strategy is to calculate tally after every vote submission and compare it with the previous tally to learn the voter's intent. Due to a correctness, Π should be able to produce meaningful election tally for any turnout, including just one voter. Since an adversary $\mathcal{A}$ controls a trustee T, he should be able to compute results after very first cast procedure is done. This means that either $\mathcal{A}$ may learn voters' choices one by one or Π fails to produce results for any subset of abstained voters and therefore is not correct.

5.1.1 EA + VC

It's not enough for privacy, an adversary can perform exactly the same attack as for 'EA only case'

5.1.2 EA + T

Private. Example: Demos

5.1.3 EA + VSD

In such case Π can be private, since it's enough to have a trusted VSD for preserving voters' privacy.

5.2 VSD only:

VSD is trusted, all other entities ($T, \mathrm{EA}, \mathrm{VC}$) are controlled by an adversary. BB is completely passive and simply posts all information received from VC. Consider the following e-voting system Π:

(1) EA generates a large random number R and commitments bases g, h. Also EA picks a random id x_i for every candidate $cand_i$. $R, g, h, \{cand_i, x_i\}$ - Π's public information.

(2) each voter submits vis VSD a perfectly hiding commitment $g^{x_i} h^{r_l}$, where r_l is a random number less than R picked by VSD.

(3) after the last voter cast his vote, VSD submits a commitment $g^{x_{abs}} h^{r_{vsd}}$, where x_{abs} corresponds to the abstain option and $r_{vsd} = R - \sum_{l=0}^{l=n} r_l$, n - number of voters.

(4) T computes the product of all hiding commitments and opens the result.

In order to break voters' privacy, an adversary has to break perfectly hiding commitments.

5.3 VC only

Impossible. An adversary can perform exactly the same attack as for 'EA only case.

5.3.1 VC + T

Due to the assumption that voters are not able to collaborate for protecting their privacy and all complex math operation performed by VSD or EA, it's impossible to have privacy in this case. A voter provides VSD with an input which is either in a plain text or calculated by EA line and therefore immediately gives away his choice.

5.3.2 VC + VSD

In such case Π can be private, since it's enough to have a trusted VSD for preserving voters' privacy.

5.3.3 VC + EA

Impossible. See the EA + VC case for details.

5.4 T only

Impossible. Same reasoning as VC + T case.

5.4.1 T + VC

seems meaningless

5.4.2 T + VSD

In such case Π can be private, since it's enough to have a trusted VSD for preserving voters' privacy.

5.4.3 T + EA

Private. See the EA + T case for details.

6 Strict privacy notion

Voter privacy implies that a voter is capable of casting his own vote secretly and freely without letting others' parties, namely an adversary, to learn some information about his preferences or interfere in it.

In general, the goal of the adversary who attacks voter privacy is to learn some information about the candidate selections of the honest voters. We define an attack against voters privacy as successful, if there is an election result*, for which an adversary is capable of distinguishing how the honest voters voted while it can observe the whole e-voting network** and corrupt some part of honest voters and some entities and also has access to honest voters' receipts.

*Obviously, it doesn't include trivial election results, where all voters voted for the same option.
**Except channels that are marked as untrappable.

In this section we use term strict privacy to imply that an adversary interacts with an e-voting system on behalf of all voters and there is no such thing as honest voters. In general, strict privacy means that even the person who cast a vote can

not break his own privacy after the **Cast** protocol is completed.

We formally define strict voter privacy via a Voter Privacy game, denoted as $G^{\mathcal{A},Sim}_{strict,<honest\ entities>}(1^\lambda)$, that is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, and takes as input the security parameter λ* and returns 1 or 0 depending on whether the adversary wins. Also, $\mathcal{A}$ is allowed to corrupt some entities. The choice of the corrupted parties splits the Voter Privacy game into two different scenarios. All meaningful* cases of collusion fall into two scenarios: (1) entities (EA, T) are hones and (2) VSD is honest.
Thomas, do you think that m and n (the number of candidates and the number of voters) should be added as the input parameters to the Voter Privacy game? Now, it's up to adversary to define an election parameters.
→Reply: good observation! It is not necessary for privacy as long as n, m are polynomial in λ, which is the case we consider usually. *In cases where only EA or T can be trusted it is not possible to have voter privacy.

According to the game rules, an adversary is allowed to define the election parameters and act on behalf of all voters and corrupted entities. During the game $\mathcal{C}$ plays the role of honest parties and returns to the adversary simulated and real view of a voter in order defined by a coin a (If $a = 0$, $\mathcal{C}$ returns $(simulated\_view, real\_view)$ and $(real\_view, simulated\_view)$ otherwise). A challenger flips the coin a only once, before interacting with $\mathcal{A}$. Note that the result of the game heavily depends on an efficiency of the simulator that is used by the challenger for producing a simulated view. →no need to write the last period. If we use complexity leveraging for privacy, the simulator is superpolynomial.

We will show that the notion of strict privacy is the weakest level of privacy that contradicts end-to-end verifiability. →Emphasise this statement more. It is your main result.

6.1 EA and T are honest: $G^{\mathcal{A},Sim}_{strict,\mathrm{EA},T}(1^\lambda)$

In the case of trusted EA and T, $\mathcal{C}$ uses a simulator Sim to generate a fake credentials so an adversary would not be able to distinguish the result of the **Cast** protocol run according to its intent and with credentials from a run with another list of candidate selections and fake credentials.

The behaviour of $\mathcal{C}$ in the $G^{\mathcal{A},Sim}_{strict,\mathrm{EA},T}(1^\lambda)$ game captures the ability of an honest voter to lie about his vote in code-based e-voting schemes. Suppose, an adversary makes a voter V to cast his vote for an option $U_\mathcal{A}$ instead of the voter's original intent U_V. V still can cast vote for the option U_V and fake his credentials in a way that his actual receipt would correspond to the option $U_\mathcal{A}$. If by faking credential V can fool an adversary into believing that he cast vote for $U_\mathcal{A}$ then his privacy is preserved.

For every vote that an adversary cast, $\mathcal{C}$ creates a fake voter who cast exactly the same vote. Every round the sum of an adversarial vote and an challenger's vote would give $U_\mathcal{A}$ + U_V. this sentence does not read well and maybe it could be removed. An actual vote distribution depends on the credentials choice. If $\mathcal{A}$ uses real credentials he votes for the $U_\mathcal{A}$ and $\mathcal{C}$ – for the U_V. Otherwise , $\mathcal{A}$' vote corresponds to the option U_V and $\mathcal{C}'$ vote to the $U_\mathcal{A}$. $\mathcal{A}$ is provided with both credentials real and fake in order defined by $\mathcal{C}$'s coin a. If $a = 0$ $\mathcal{C}$ returns $(fake\_credentials, real\_credentials)$ and $(real\_credentials, fake\_credentials)$ otherwise. An adversary tries to guess the coin a. $\mathcal{A}$ may stop the game in any moment *.
Not really. An adversary can not stop the game in the moment after he cast his vote and before $\mathcal{C}$ posts a fake voter's vote.

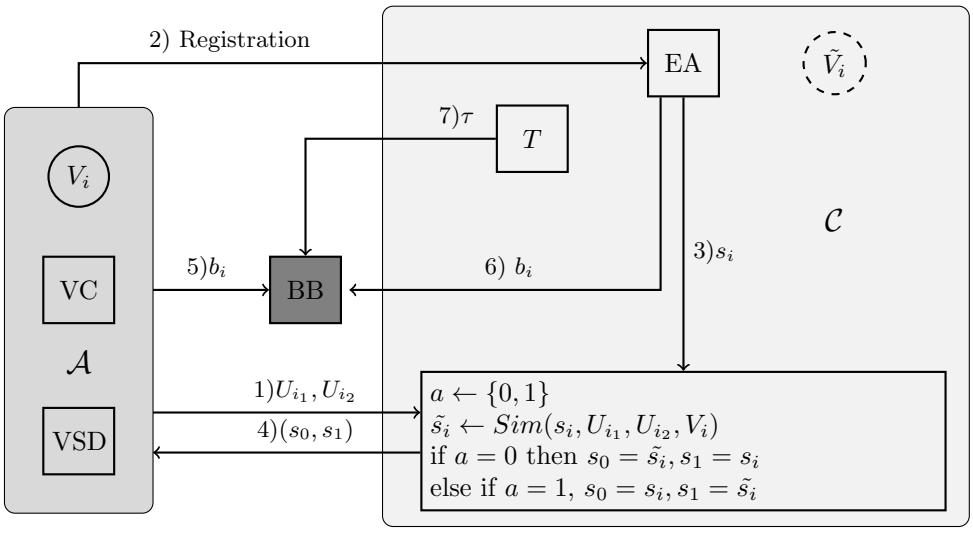


FIGURE 4: $G^{\mathcal{A},Sim}_{strict,\mathrm{EA},T}(1^\lambda)$

In the game $G^{\mathcal{A},Sim}_{strict,\mathrm{EA},T}(1^\lambda)$, an adversary $\mathcal{A}$ interacts with the challenger $\mathcal{C}$ on behalf of all voters, VC and VSD. $\mathcal{C}$ plays the role of EA and T. BB is completely passive and represents a publicly viewed database.

After the adversary $\mathcal{A}$ defines the election parameters, $\mathcal{C}$ doubles the voters, computes setup data and starts the election. Thomas: Can the adversary create malicious parameters? For example what if it creates trapdoors in the parameter generation like NSA probably does?. This duplication of voters is needed because $\mathcal{C}$ would post another ballot for every adversarial's cast ballot. If $\mathcal{A}$ abstains, $\mathcal{C}$ abstains as well. During the game, we will refer to duplicated voters as fake ones.

$\mathcal{A}$ 1) picks and sends two options U_{i_1}, U_{i_2} * to the challenger $\mathcal{C}$. One of the options U_{i_1}, U_{i_2} is his intent, the other – the option that the challenger would use in order to produce an indistinguishable from the intent's ballot and receipt view*. After sending options, 2) $\mathcal{A}$ runs the **Registration** as if he is V_i. $\mathcal{C}$ creates a fake credentials $\tilde{s_i}$ and 3) generates real credentials s_i for him. 4) $\mathcal{C}$ responds $\mathcal{A}$ with a pair of credentials s_0, s_1, where one of the credentials are real and the other were generated using the simulator Sim in a such way, that no matter what credential s_0 or s_1 $\mathcal{A}$ uses to cast a vote for U_{i_1}, the produced ballot would also correspond to the result of the **Cast** protocol for an option U_{i_2} with the other credentials and vice versa. If 5) $\mathcal{A}$ choses to post the ballot b_i to BB, 6) $\mathcal{C}$ posts exactly the same ballot to BB on behalf of the fake voter. When $\mathcal{A}$ stops the election, 7) $\mathcal{C}$ posts the tally τ **.

*Haven't thought about the case when $U_{i_1} = U_{i_2}$. Should the challenger return same credentials or Sim would generate something different? On the one hand, it's fine for a Sim to generate exactly the same credentials. If it generates something different, the result of the game depends on how good Sim is. On the other hand, case $U_{i_1} = U_{i_2}$ is a bit weird. This game tries to capture a way how a honest voter fakes his credentials to convince $\mathcal{A}$ that he voted as he was told. There is no point in faking anything if you may as well give back real stuff.
Thomas: There is no reason to restrict the adversary from submitting $U_{i_1} = U_{i_2}$. It is just a stupid adversary that will not probably succeed... Remarks:
*If $\mathcal{C}$ succeeds then $\mathcal{A}$ wouldn't be able to say whether its ballot and receipt corresponds to U_{i_1} or U_{i_2} and BB would contain both ballots (one for the option U_{i_1} ,the other for the option U_{i_2}) so the tally wouldn't reveal any information. Example: if $\mathcal{A}$ picks real credential for the voter V_i then it casts a vote for the option U_{i_1} ,then at the same time exactly the same ballot posted by $\mathcal{C}$ on behalf of the

fake voter $\tilde{V}_i$ would correspond to the fake credentials and the option U_{i_2}. In this case $\mathcal{A}$ voted for U_{i_1}. Else if $\mathcal{A}$ picks fake credentials and tries to vote for the option U_{i_1}, his ballot corresponds to the real credentials and the option U_{i_1}. At the same time, exactly the same ballot posted by $\mathcal{C}$ on behalf of the fake voter $\tilde{V}_i$ would correspond to the fake credentials and the option U_{i_1}. So $\mathcal{A}$ voted for U_{i_1}. If this definition holds, $\mathcal{A}$ has no idea whom he voted for. Can you maybe edit this remark? It is hard to understand...

** The tally τ is posted only if all ballots are correctly formed: were produced with one of the provided credentials, for the candidate U_{i_1}, for every cast ballot there is a duplicate, etc.
Thomas, may be you are right and here I should use two sets of votes that sums to the same tally. I'm not entirely sure, though. This approach seems quite similar to the 'only VSD is trusted' case. Here $\mathcal{A}$ can't say wether he voted for U_{i_1} and $\mathcal{C}$ for U_{i_2} or vice versa. On the one hand, he knows options U_{i_1} and U_{i_2} and just couldn't link them with voters. U_{i_1} and U_{i_2} are defined by $\mathcal{A}$ and keep changing during the game. Generally, in order to save my privacy, I should be able to lie about my intent. Which means that if adversary told me to vote for U_{i_1} and I voted for U_{i_2} instead, he should not detect the lie. He even gets less information, since I definitely not going to give him 2 views. This definition addresses exactly this type of lying, however it's not how you defined privacy in Demos. Here $\mathcal{A}$ knows that both options in BB and knows which entry corresponds to v_i and which one to $\tilde{V}_i$, in Demos only one ballot is posted. Is lying about just one pair is enough? $\mathcal{A}$ himself is not sure wether he voted for U_{i_1} or for U_{i_2} (if he picks the real credentials then it's U_{i_1}, otherwise – U_{i_2}). Here, I've tried to give an adversary as many information and freedom to operate as possible: picking options adaptively instead of fixed set and receiving 2 views instead of just a single one. Does anything here make sense?

$G^{\mathcal{A},Sim}_{strict,\mathrm{EA},T}(1^\lambda)$ defined as follows:

(1) During the game $\mathcal{C}$ plays the role of EA and T. $\mathcal{A}$ operates on behalf of the all voters, VSD and VC.

(2) $\mathcal{A}$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$, a set of allowed candidates' selections $\mathcal{U}$. It provides $\mathcal{C}$ with $\mathcal{V}, \mathcal{P}, \mathcal{U}$.

(3) $\mathcal{C}$ doubles the set $\mathcal{V}$ adding fake voters $\{V'_1, ..., V'_n\}$ and starts the election on

behalf of EA. Also, $\mathcal{C}$ flips a coin $a \leftarrow \{0,1\}$ to define an order according to which real and simulated credentials would be returned to $\mathcal{A}$.

(4) The adversary $\mathcal{A}$ picks two option $U_{i_1}, U_{i_2} \in \mathcal{U}$, where U_{i_1} is an option for the real credentials and U_{i_2} is an option for the fake ones. After that, $\mathcal{A}$ and $\mathcal{C}$ engage in an interaction where $\mathcal{A}$ runs the **Registration** protocols on behalf of all voters. For each voter $V_i \in \mathcal{V}$:

– $\mathcal{C}$ picks a fake voter $\tilde{V}_i$ and generates credentials $\tilde{s}_i$ for him using Sim*. $\mathcal{C}$ responds $\mathcal{A}$ with a pair of simulated and real credentials (s_0, s_1) in order defined by the coin a:
$$\begin{cases} \text{if } \ a = 0, \ \ (s_0, s_1) = (\tilde{s}_i, s_i) \\ \text{else } \ (s_0, s_1) = (s_i, \tilde{s}_i) \end{cases}$$

– Using one of the credentials $\mathcal{A}$ schedules the **Cast** protocol executions to vote for an option U_{i_1} and sends the produced ballot b to $\mathcal{C}$. If $\mathcal{A}$ used the real credentials, then the obtained ballot b indeed contains a vote for an option U_{i_1}. Otherwise, ballot b was generated for the fake credentials and in reality it corresponds to the option U_{i_2}. Since $\mathcal{A}$'s ballots are always tallied by $\mathcal{C}$ as if real credentials were used, ballot b would be a vote for an option U_{i_1} only if $\mathcal{A}$ picks the real credentials.

– If $\mathcal{A}$ posts a ballot to BB, $\mathcal{C}$ posts exactly the same ballot in the entry that corresponds to the fake voter $\tilde{V}_i$. Since $\mathcal{C}$'s ballots are tallied with respect to the fake credentials, the ballot produced by $\mathcal{A}$ in this case would correspond to the remaining option. So if $\mathcal{A}$ piked the real credentials, $\mathcal{C}$'s ballot is for the vote U_{i_1}, and for the vote U_{i_2} otherwise. BB contains $\mathcal{A}$'s and $\mathcal{C}$'s ballots and the resulting Tally is the summ of all options U_{i_1} and U_{i_2} picked by $\mathcal{A}$

(5) $\mathcal{C}$ executes the $Tally$ protocol.

(6) Finally, $\mathcal{A}$ using all information collected above (including the contents of the BB) outputs a bit a^*

(7) The game returns a bit which is 1 if $a = a^*$ and 0 otherwise.

***Remark:**
Sim works in a way that the fake credentials satisfy both of the following rules:

(1) The real credentials and U_{i_1} option should give ballot and receipt, which are identical* to ballot and receipt produced for the fake credentials and U_{i_2} option.

(2) The fake credentials and U_{i_1} option should give ballot and receipt, which are identical to ballot and receipt produced for the real credentials and U_{i_2} option.

* indistinguishable ?

For understanding logic behind the privacy definition defined above, consider the following toy example:
There are just 3 voters and 3 possible options op_1, op_2, op_3. Challenger's coin $a = 0$, so the fake credentials are always would be the first credentials in the credentials pair. Suppose $\mathcal{A}$' schedules three vote casting protocols:
1) $U_{i_1} = op_3, U_{i_2} = op_1$ and picks the second credentials for generating the ballot b_1 for op_3. Since $\mathcal{A}$ picked the fake credentials, the ballot b_1 in the Tallying process would result in a vote for op_1. Meanwhile, $\mathcal{C}$ posts identical to the ballot b_1 ballot $\tilde{b_1}$ that corresponds to an option op_3. 2) $U_{i_1} = op_1, U_{i_2} = op_2$ and picks the first credentials for generating the ballot b_2 for op_1. Since $\mathcal{A}$ picked the real credentials, the ballot b_2 in the Tallying process would correctly result in a vote for op_1. Meanwhile, $\mathcal{C}$ posts identical to the ballot b_2 ballot $\tilde{b_2}$ that corresponds to an option op_2. 2) $U_{i_1} = op_3, U_{i_2} = op_1$ and picks the first credentials for generating the ballot b_3 for op_3. Since $\mathcal{A}$ picked the fake credentials, the ballot b_2 in the Tallying process would result in a vote for op_1. Meanwhile, $\mathcal{C}$ posts identical to the ballot b_3 ballot $\tilde{b_3}$ that corresponds to an option op_3. At the end of this mini election, $\mathcal{A}$ would get the following result $op_1 = 3, op_2 = 1, op_3 = 2$.
Even though $\mathcal{A}$ intended to vote for options op_3, op_1, op_3, his real choices are op_1, op_1, op_1. If privacy holds, $\mathcal{A}$ is unable to understand whether he voted for U_{i_1} and the challenger for an option U_{i_2} or vice versa.

Strict privacy: EA **and** T **are honest:**

The e-voting system Π achieves strict voter privacy in case of trusted EA and T, if there is a PPT simulator Sim such that for any PPT adversary $\mathcal{A}$:

$$|\Pr[G^{\mathcal{A},Sim}_{strict,\mathrm{EA},T}(1^\lambda) = 1] - \tfrac{1}{2}| = negl(\lambda)$$

6.2 VSD is honest: $G^{\mathcal{A},Sim}_{strict,\mathrm{VSD}}(1^\lambda)$

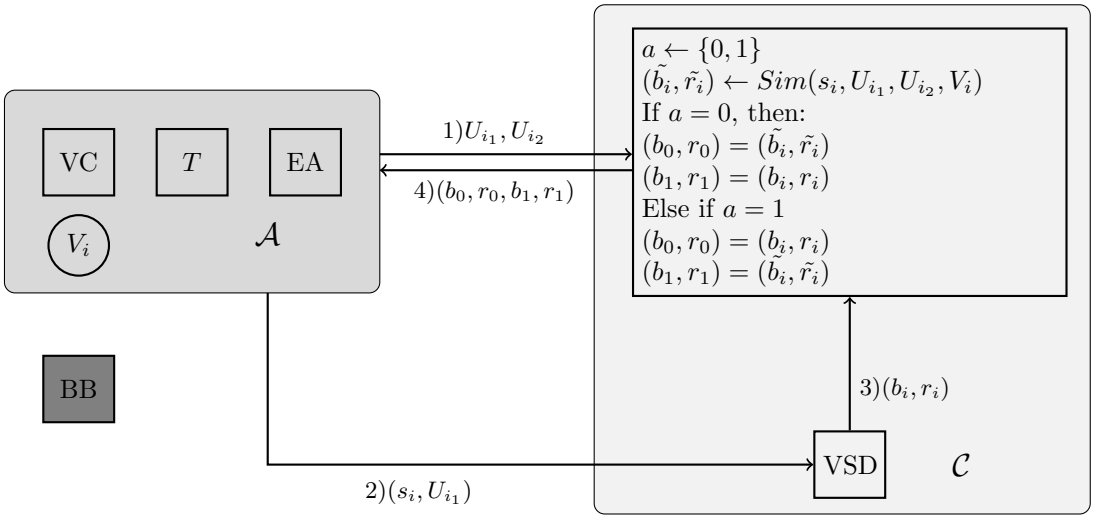


FIGURE 5: $G^{\mathcal{A},Sim}_{strict,\mathrm{VSD}}(1^\lambda)$

In the game defined above, $\mathcal{A}$ operates on behalf of the all voters and all corrupted entities, such as: EA, VC and T. BB is completely passive and represents a publicly accessible database.
1) an adversary $\mathcal{A}$ picks and sends options U_{i_1}, U_{i_2} to the challenger $\mathcal{C}$. After that 2) $\mathcal{A}$ schedules the **Cast** protocol with $\mathcal{C}$ as if he is a voter V_i. $\mathcal{C}$ 3) generates a real ballot and receipt and uses Sim to create a fake ones. At the end 4) $\mathcal{C}$ responses with a pair of ballots and receipts b_0, r_0, b_1, r_1, where one ballot and receipt corresponds to an option U_{i_1}, the other was generated using the simulator Sim for an option U_{i_2}.

The game $G^{\mathcal{A},Sim}_{strict,\mathrm{VSD}}(1^\lambda)$ is defined as follows:

(1) During the game $\mathcal{C}$ plays the role of VSD. $\mathcal{A}$ operates on behalf of the all voters and all corrupted entities, such as: EA, VC and T.

(2) $\mathcal{A}$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$, a set of allowed candidates' selections $\mathcal{U}$ and starts the election.

(3) $\mathcal{C}$ flips a coin $a \leftarrow \{0,1\}$ to define an order according to which real and simulated ballots and receipts would be returned to $\mathcal{A}$.

(4) The adversary A and the challenger $\mathcal{C}$ engages in an interaction where $\mathcal{A}$ runs the Cast protocols on behalf of all voters. For each voter $V_i \in \mathcal{V}$:

> – $\mathcal{A}$ sends to $\mathcal{C}$ options U_{i_1}, U_{i_2} and starts with $\mathcal{C}$ the **Cast** protocol on behalf of V_i. As a result of the protocol execution $\mathcal{C}$ provides $\mathcal{A}$ with a pair of simulated and real ballot and receipt $(b_0, r_0)(b_1, r_1)$ s.t.:
> $\begin{cases} \text{if } \ a = 0, \ \ (b_0, r_0) = (\tilde{b_i}, \tilde{r_i}) \ \text{ and } \ (b_1, r_1) = (b_i, r_i) \\ \text{else } \ (b_0, r_0) = (b_i, r_i) \ \text{ and } \ (b_1, r_1) = (\tilde{b_i}, \tilde{r_i}) \end{cases}$
> where the pair (b_i, r_i) is the ballot and receipt for an adversarial option U_{i_1} and $(\tilde{b_i}, \tilde{r_i})$ is the ballot and receipt for U_{i_2} option generated via the simulator Sim.

(5) Finally, $\mathcal{A}$ executes the $Tally$ protocol and using all information collected above outputs a bit a^*

(6) The game returns a bit which is 1 if $a = a^*$ and 0 otherwise

Strict privacy: VSD **is honest:**

The e-voting system Π achieves strict voter privacy in case when VSD is trusted, if there is a PPT simulator Sim such that for any PPT adversary $\mathcal{A}$:

$|\Pr[G^{\mathcal{A},Sim}_{strict,\mathrm{VSD}}(1^\lambda) = 1] - \frac{1}{2}| = negl(\lambda)$

7 Strict privacy vs E2E Verifiability

Any system Π that satisfies the definition of strict privacy, is "receipt free" but not E2E Verifiable.
The Real and Ideal executions in this case are shown on the figure 6.
Security requires existence of some simulator program $\mathcal{S}$ such that the real and ideal executions are indistinguishable for and any environment $\mathcal{Z}$. Unfortunately, as we are about to show, no such simulator exists.

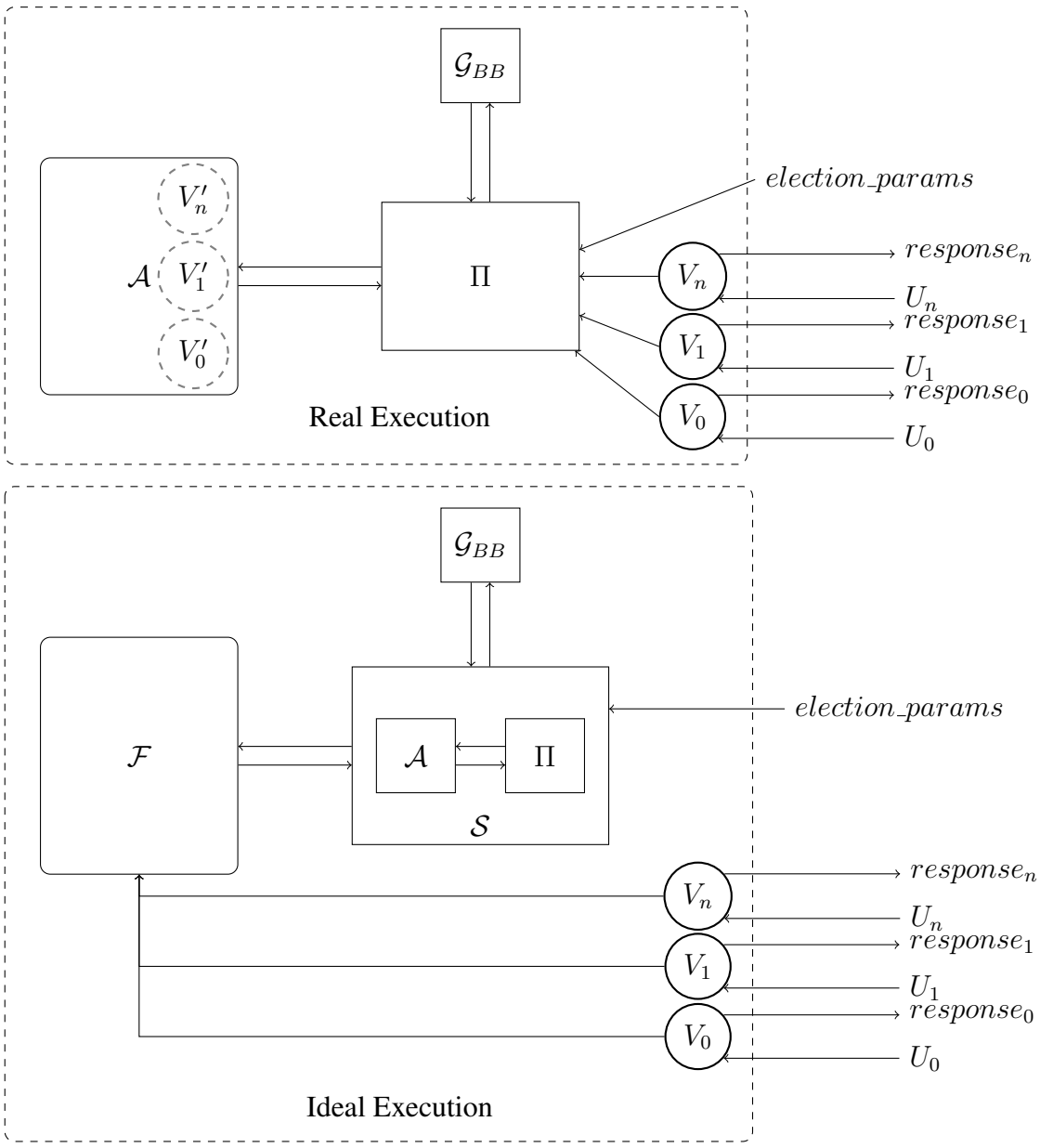


FIGURE 6: Real and Ideal Execution

Proposition:
For any system Π that satisfies the definition of strict privacy, there is an adversary $\mathcal{A}$ such that for any simulator $\mathcal{S}$, there is an environment $\mathcal{Z}$ that distinguishes ideal and real executions $\mathrm{EXEC}^{\mathcal{F},\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{S}} \not\approx \mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}}$

7.1 Proof: part 1

Suppose we have a system Π, which is strictly private in case "EA is corrupted, but VSD is honest".
Consider the following attack against E2E Verifiability:

$\mathcal{A}$: – corrupts EA and VSDs but doesn't corrupt ASDs. – corrupts t voters $(t \leq n)$, where n is the total number of voters. – creates fake voters $\{V'_0, V'_1, \ldots, V'_n\}$ and using its power substitutes some part γ of honest voters with fake ones for the U_a option. – fake voters vote for adversarial options according to a vote-casting procedure. – substituted honest voters receive receipts generated for fake voters.

let $\mathcal{Z}$ be the environment that works as follows:

$\mathcal{Z}$: – defines an election setup information: $election\_params = (\mathcal{C}, \mathcal{V}, \mathcal{U}, params)$, where $\mathcal{C}$ - list of candidates, $\mathcal{V}$ - list of voters, $\mathcal{U}$ - list of allowed candidates' selections, $params$ - other required information. – instructs each voter V_i to vote for the blank '$no\ one$' option. – stops the vote-casting phase. – asks $\mathcal{G}_{BB}$ for the election result τ – asks every voter V_i to verify his choice and return the result of verification - $response_i$, where $response_i$ equal to $(sid, verify\_responce, \tau')$ in case of successful verification and $\perp$ otherwise. – If the number of successful verification responses equal to the number of voters and in all responses provided by a voter tally τ' is equal to the $\mathcal{G}_{BB}$'s tally τ return 1. Otherwise return 0.

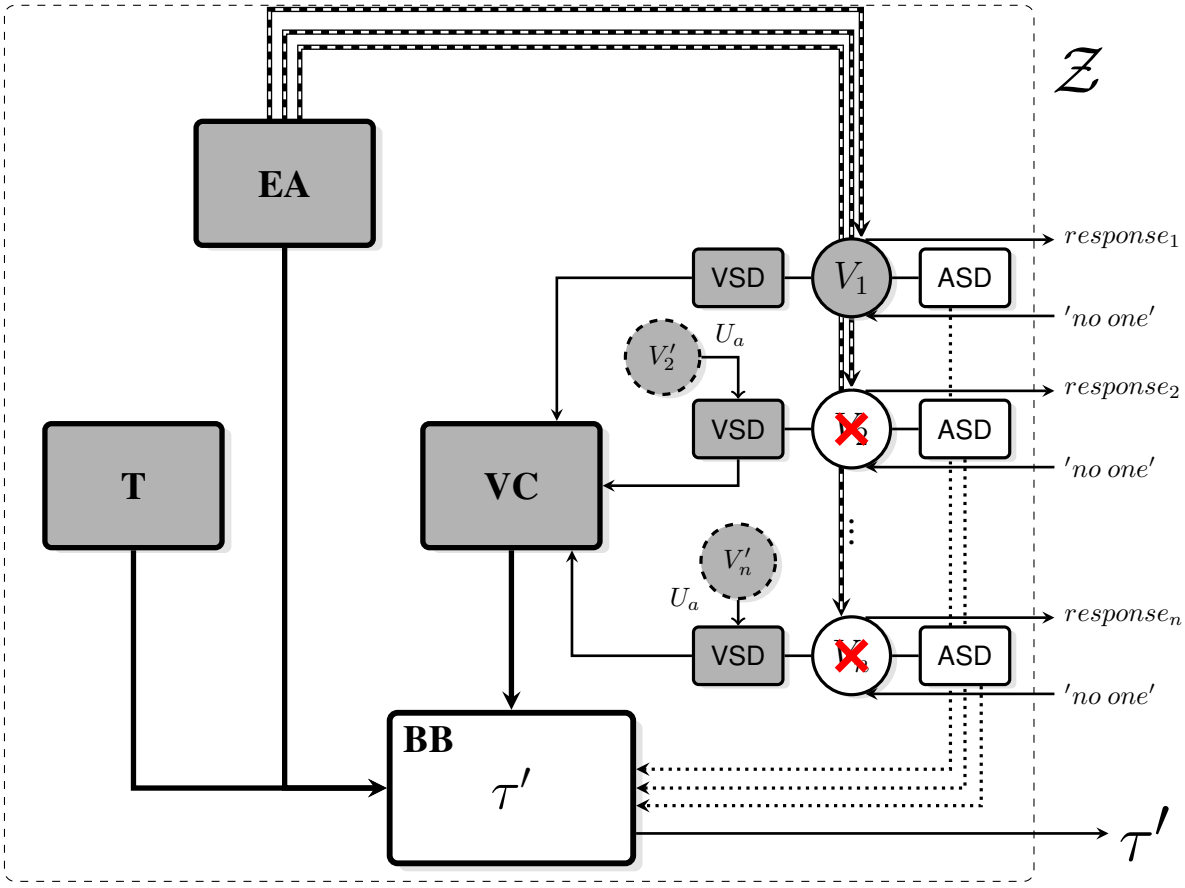


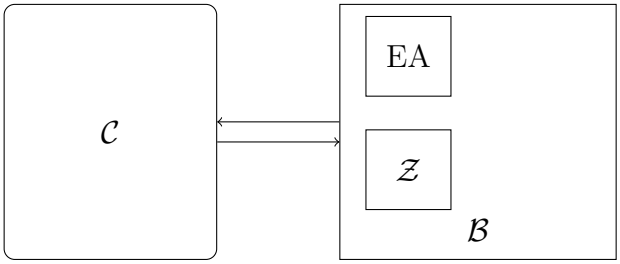
FIGURE 7: $\mathcal{A}$

Since Π is strictly private, probability, that an adversary $\mathcal{D}$ engaging in the **Cast** protocol would distinguish real and simulated view and win the attack against the strict privacy $Pr[G^{\mathcal{D},Sim}_{strict,\mathrm{VSD},T}(1^\lambda) = 1] = \frac{1}{2} + \alpha$ where α is negligible. In case of corrupted EA, $\mathcal{C}$ returns a real ballot and receipt pair and a simulated via Sim pair. This means that in a strictly private system an adversary has a negligible chance to distinguish the ballot and receipt for his option and for some U_c option.

During the Real Execution a voter can either accept or reject the cast ballot. Rejection is possible only if he detects that the receipt and ballot are faked. The probability that a voter distinguish the receipt and ballot for his option from for the 'no-one' option and rejects is $\Pr[V_i\ rejects] \leq \alpha$, where α is negligible.

By nature of the attack $\mathcal{A}$, fake receipt is the perfectly valid receipt for the U_a option and the cast ballot would always be successfully verified. Therefore in the Real Execution $\mathcal{Z}$ would get a successful verification response from all n voters and output 1 with the probability $\Pr[\text{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 1] = \Pr[all\ V_i\ accept] = \prod_{i=0}^{n}(1 - \Pr[V_i\ rejects]) \geq (1-\alpha)^n = 1 - n\alpha = 1 - \beta$, where $\beta = n\alpha$ is negligible. Thus, $\Pr[\text{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] = 1 - \Pr[\text{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 1] \leq \beta$, where β is negligible.

Suppose for the sake of contradiction that $\Pr[\text{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] \geq \beta$, where β is non-negligible. This means that at least one voter rejects the receipt. I will show that this contradicts the definition of strict privacy. Consider an attacker $\mathcal{B}$ against strict privacy which exploits the environment $\mathcal{Z}$.



$\mathcal{B}$ interacts with the challenger in the strict privacy attack $\mathcal{C}$ as follows:
– $\mathcal{Z}$ defines an election parameters $election\_params$, assign every voter to vote for the blank option $\{V_i, `no\ one'\}$ and send this information to the $\mathcal{B}$.
– $\mathcal{B}$ forwards received $election\_params$ to the EA.
– $\mathcal{B}$ corrupts the EA.
– EA starts the election.
– $\mathcal{B}$ interacts with the challenger $\mathcal{C}$ provides $(V_i, `no\ one', U_{\mathcal{Z}_i})$ as the input.
– $\mathcal{C}$ sends back to the $\mathcal{B}$ real and simulated view $(b_0, r_0), (b_1, r_1)$ in the order defined by the challenger's coin a.
– $\mathcal{B}$ votes on behalf of all voters
– $\mathcal{Z}$ stops the vote-casting phase.
– B computes the election's tally and proof of the tally's correctness.
– $\mathcal{Z}$ asks $\mathcal{G}_{BB}$ for the election result τ
– $\mathcal{Z}$ requests every voter to verify his ballot correctness.
– $\mathcal{B}$ will run the verification on behalf of all voters using ballots and receipts from $\{b_0, r_0\}$.
– $\mathcal{Z}$ outputs 1 or 0 depending on the voters' verdict.
–$\mathcal{B}$ outputs whatever the $\mathcal{Z}$ outputs.

The challenger $\mathcal{C}$ outputs simulated ballot and receipt as (b_0, r_0), when the coin $a = 0$ and as (b_1, r_1) otherwise.

In case when $a = 0, \mathcal{B}$'s behaviour is identical to the $\mathcal{A}$'s strategy and $\mathcal{B}$ wins if $\mathcal{Z}$ outputs 0, which happens if at least one voter rejects the simulated receipt. By assumption $\Pr[\text{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] \geq \beta$, where β is non-negligible. Therefore if (b_0, r_0) is the set of simulated ballot and receipt, $\mathcal{B}$ wins with the probability $Pr[\mathcal{B} \to 0 | a = 0] = \Pr[\text{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] \geq \beta$, where β is non-negligible.

On the other hand, when $a = 1$, $\mathcal{B}$ plays honestly and the probability of $\mathcal{Z}$ outputting 1 is equal to probability that all voters successfully verify their votes in the honest execution, which is happens with overwhelming probability $1 - negl(\lambda)$.

The probability of $\mathcal{B}$ winning the attack against the strict privacy is $\Pr[G^{\mathcal{B}}_{strict}(1^\lambda) = 1] = \Pr[\mathcal{B} \to 0 | a = 0] \Pr[a = 0] + \Pr[\mathcal{B} \to 1 | a = 1] \Pr[a = 1] = \Pr[\text{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] \Pr[a = 0] + \Pr[\text{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},honest} = 1] \Pr[a = 1] \geq \frac{1}{2}\beta + \frac{1}{2} - negl(\lambda)$, where β is not negligible. This implies that $\mathcal{B}$ wins the attack against strict privacy with the probability more than $\frac{1}{2} + negl(\lambda)$, which contradicts the assumption that the Π is strictly private.

In the Ideal Execution any simulator S can either:
1) post in $\mathcal{G}_{BB}$ the tally τ' generated by $\mathcal{A}$ or any other tally $\tau' \neq \tau$
or
2) ignore the $\mathcal{A}$'s tally and post the actual tally τ.

In the first case, the ideal functionality for E2E verifiability $\mathcal{F}$ would always detect the tally deviation caused by $\mathcal{A}$ if such exists. And since $\mathcal{A}$ doesn't corrupt ASDs, for all honest voters the ideal functionality $\mathcal{F}$ would block verification responses. This implies that in the Ideal Execution $\mathrm{EXEC}^{\mathcal{F},\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{S}}$ $\mathcal{Z}$ would get no response from honest voters. The total number of successful verifications would be equal to the number of corrupted voters, which is less (if not voters are corrupted) than the total number of voters – $\mathcal{Z}$ outputs 0.

In the second case, there exists a class of simulators which ignore $\mathcal{A}'$ actions and post the actual tally. For those simulators consider an modified environment $\tilde{\mathcal{Z}}$ that works as follows:

$\tilde{\mathcal{Z}}$:
Outputs a bit according to the following rules:
$$\begin{cases} \text{if } \mathcal{Z} \text{ outputs 1 and the number of non-blank votes greater or equal } \gamma\text{– output 1} \\ \text{else output 0} \end{cases}.$$

$\tilde{\mathcal{Z}}$ would still output 1 in case of the real execution since the number of non-blank votes would be at least γ due to successful attack $\mathcal{A}$. However $\tilde{\mathcal{Z}}$ would not find at least γ non-blank votes and output 0 in the ideal execution.

Thus, there is the attacker $\mathcal{A}$ such that for any simulator $\mathcal{S}$ there is the environment $\tilde{\mathcal{Z}}$ or $\mathcal{Z}$ which can always distinguish real and ideal executions.

7.2 Proof: part 2

Suppose we have a system Π', which is strictly private in case "EA is honest, but VSD is corrupted".
Consider the following attack against E2E Verifiability:

$\mathcal{A}'$:
– corrupts EA and VSDs but doesn't corrupt ASDs.
– chooses an option $U_a \in \mathcal{U}$
– corrupts t voters $(t \leq n)$, where n is the total number of voters.
– provides every honest voter V_i with fake credentials s'_i s.t. the ballot and receipt produced using $(s'_i, `no\ one')$ are identical to the ballot and receipt produced using real credentials for an option $U_a$: $b, r \leftarrow (s'_i, `no\ one')$ AND $b, r \leftarrow (s_i, U_a)$
– creates fake voters $\{V'_0, V'_1, \ldots, V'_n\}$ and provides them with real credentials s_i
– using its power substitutes some part γ of honest voters' cast protocols with the fake voters' protocols for the U_a option.

let $\mathcal{Z}'$ be the environment that works as follows:

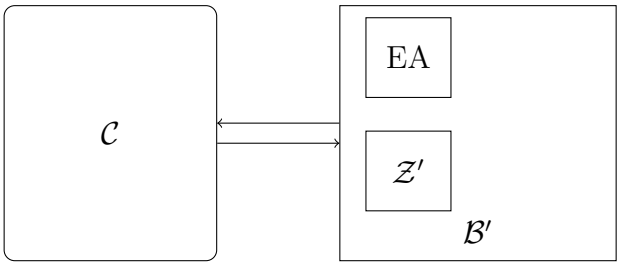
$\mathcal{Z}'$:
– defines an election setup information:
$election\_params = (\mathcal{C}, \mathcal{V}, \mathcal{U}, params)$, where $\mathcal{C}$ - list of candidates, $\mathcal{V}$ - list of voters, $\mathcal{U}$ - list of allowed candidates' selections, $params$ - other required information.
– instructs each voter V_i to vote for the blank option ('no one').
– stops the vote-casting phase.
– asks $\mathcal{G}_{BB}$ for the election result τ
– asks every voter V_i to verify his choice and return the result of verification - $response_i$, where $response_i$ equal to $(sid, verify\_responce, \tau')$ in case of successful verification and $\perp$ otherwise.
– If the number of successful verification responses equal to the number of voters and in all responses provided by a voter tally τ' is equal to the $\mathcal{G}_{BB}$'s tally τ return 1. Otherwise return 0.

Since Π' is strictly private, probability, that an adversary engaging in the **Registration** protocol would distinguish real and simulated view and win the attack against the strict privacy $|\Pr[G^{\mathcal{A},Sim}_{strict,\mathrm{EA}}(1^\lambda) = 1] - \frac{1}{2}| = \alpha$ where α is negligible. In case "EA is honest, but VSD is corrupted", simulated view is credentials $\tilde{s_i}$ generated via the simulator Sim.

In the Real Execution a voter V_i starts the Cast protocol using fake credentials s'_i and an option $`no\ one'$. However, $\mathcal{A}'$ substitutes his ballot and receipt with the ballot and receipt generated for a fake voter with real credentials and U_a op-

tion. By nature of the attack, the returned receipt generated for real credentials and the U_a option is identical to the receipt produced for a fake credentials and '$no\ one'$ option. Therefore in the Real Execution $\mathcal{Z}'$ would output 1 if non of the voters would detect that he was given a receipt for fake credentials. The probability of this event is $\Pr[\text{EXEC}^{\Pi',\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{A}'} = 1] = \Pr[all\ V_i\ accept] = \prod_{i=0}^{n}(1 - \Pr[V_i\ rejects])$. Suppose, the probability of rejection by V_i is $\Pr[V_i\ rejects] = \zeta$. Thus, $\Pr[\text{EXEC}^{\Pi',\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{A}'} = 1] = (1-\zeta)^n = 1 - n\zeta = 1 - \zeta'$.

Suppose that ζ' is not negligible. This means that $\Pr[\text{EXEC}^{\Pi',\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{A}'} = 0] = 1 - \Pr[\text{EXEC}^{\Pi',\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{A}'} = 1] = \zeta'$, where ζ' is not negligible. This means that at least one voter rejects the receipt with a non-negligible probability. We will show that this contradicts the definition of strict privacy. Consider an attacker $\mathcal{B}'$ against strict privacy which exploits the environment $\mathcal{Z}'$.



$\mathcal{B}'$ interacts with the challenger in the strict privacy attack $\mathcal{C}$ as follows:
– $\mathcal{Z}$ defines an election parameters $election\_params$, assign each voter the blank option $\{V_i, `no\ one'\}$ and send this information to the $\mathcal{B}$.
– $\mathcal{B}'$ forwards received $election\_params$ to the $\mathcal{C}$.
– $\mathcal{B}'$ corrupts the VSD.
– $\mathcal{C}$ starts the election.
– $\mathcal{B}'$ sends to $\mathcal{C}$ $(V_i, `no\ one', U_a)$.
– $\mathcal{C}$ sends back s_0, s_1.
– $\mathcal{B}'$ provides voters with credentials s_0 and uses $(s_0, `no\ one')$ to produce the ballot and receipt and posts the result on BB.
– $\mathcal{Z}'$ stops the vote-casting phase.
– $\mathcal{C}$ executes the $Tally$ protocol.
– $\mathcal{Z}'$ asks $\mathcal{G}_{BB}$ for the election result τ
– $\mathcal{Z}'$ requests every voter to verify his ballot correctness.
– $\mathcal{B}'$ will run the verification on behalf of all voters.
– $\mathcal{Z}'$ outputs 1 or 0 depending on the voters' verdict.
– $\mathcal{B}'$ outputs whatever the $\mathcal{Z}'$ outputs.

In case when the simulated credentials are chosen ($a = 0$), $\mathcal{B}'$ provides a voter with fake credentials and generated ballot and receipt for the blank option using the fake credentials, which means that real credentials correspond to the U_a option. $\mathcal{B}'$'s behaviour is identical to the $\mathcal{A}'$'s strategy. $\mathcal{B}'$ wins if outputs 0, which happens if $\mathcal{Z}'$ outputs 0. $\Pr[\mathcal{B}' \to 0 | a = 0] = \Pr[\mathrm{EXEC}^{\Pi', \mathcal{G}_{BB}}_{\mathcal{Z}', \mathcal{A}'} = 0] = \zeta'$

Else if $a = 1$, $\mathcal{B}'$ provides a voter with real credentials and uses the real credentials to vote for the blank option, which is honest behaviour. $\Pr[B' \to 1 | a = 1] = \Pr[\mathrm{EXEC}^{\Pi', \mathcal{G}_{BB}}_{\mathcal{Z}', honest} = 1] = 1$

Thus, the probability of $\mathcal{B}'$ winning the attack against the strict privacy is $\Pr[G^{\mathcal{B}, Sim}_{strict, \mathrm{EA}}(1^\lambda) = 1] = \frac{1}{2}(\Pr[B' \to 0 | a = 0] + \Pr[B' \to 1 | a = 1]) = \frac{1}{2}(\zeta' + 1) = \frac{1}{2} + \frac{1}{2}\zeta'$, where ζ' is not negligible. This implies that $\mathcal{B}'$ wins the attack against strict privacy with the probability more than $\frac{1}{2} + negl(\lambda)$, which contradicts the assumption that the Π is strictly private.

In the Ideal Execution any simulator S can either:
1) post in $\mathcal{G}_{BB}$ the tally τ' generated by $\mathcal{A}'$ or any other tally $\tau' \neq \tau$
or

2) ignore the $\mathcal{A}'$'s tally and post the actual tally τ.

In the first case, the ideal functionality for E2E verifiability $\mathcal{F}$ would always detect the tally deviation caused by $\mathcal{A}'$ if such exists. And since $\mathcal{A}'$ doesn't corrupt ASDs, for all honest voters the ideal functionality $\mathcal{F}$ would block verification responses. This implies that in the Ideal Execution $\mathrm{EXEC}^{\mathcal{F},\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{S}}\ \mathcal{Z}'$ would get no response from honest voters. The total number of successful verifications would be equal to the number of corrupted voters, which is less (if not voters are corrupted) than the total number of voters – $\mathcal{Z}'$ outputs 0.

In the second case, there exists a class of simulators which ignore $\mathcal{A}'$ actions and post the actual tally. For those simulators consider an modified environment $\tilde{\mathcal{Z}}'$ that works as follows:

$\tilde{\mathcal{Z}}'$:
Outputs a bit according to the following rules:
$\begin{cases} \text{if } \mathcal{Z} \text{ outputs 1 and the number of non-blank votes greater or equal } \gamma\text{– output 1} \\ \text{else output 0} \end{cases}$.

$\tilde{\mathcal{Z}}'$ would still output 1 in case of the real execution since the number of non-blank votes would be at least γ due to successful attack $\mathcal{A}$. However $\tilde{\mathcal{Z}}'$ would not find at least γ non-blank votes and output 0 in the ideal execution.

Thus, there is the attacker $\mathcal{A}'$ such that for any simulator $\mathcal{S}$ there is the environment $\tilde{\mathcal{Z}}'$ or $\mathcal{Z}'$ which can always distinguish real and ideal executions.

8 Blind signature e-voting scheme

8.1 Presentation of the blind signature e-voting scheme

Our system is a two-server web-based system. The first server - *Signing server* is used for **Registration** process and the second one - VC for vote-casting. In an election, the *Signing server* mainly plays the roles of the EA but may aid the other parties. Trustee T is a separate entity that has a secure channel to communicate with EA, security of communication is supported by HTTPS. All the other parties are realized by Javascripts running at the client side. The system uses additive ElGamal encryption, so the tally is done homomorphically. Currently the blind signature scheme supports approval type of voting s.t. x-out-of-m type of option

selection, where x is between 0 and m.

8.2 Election process

The electronic voting procedure consists of 4 stages:

(1) Preparation: EA generates public pre-election data and posts it on the BB. Also EA creates and distributes envelops with private voters' information among all eligible voters. Meanwhile T obtains secret data that would be used for producing the result and posts its own pre-election data to the BB.

(2) Registration: A voter and EA engaged in an interaction during which the voter provides to the EA his private information from the envelop and proofs his identity and obtains credentials for vote casting.

(3) Vote casting: An voter starts an interaction with VSD using his credentials as proof of his eligibility to cast a vote for preferred candidates. Upon successful termination, VC receives a ballot from VSD and posts it on the BB.

(4) Decrypting: After election is closed, sum of all valid ballots is computed and decrypted by T.

8.3 Setup and parameters

Throughout the paper, we assume that we have a group of n voters $\mathcal{V} = \{V_1, \ldots, V_n\}$ that can choose as many candidates as they like among the set of m candidates $\mathcal{P} = \{P_1, \ldots, P_m\}$, n and m are both thought as polynomial functions of security parameter λ. Also, during the vote casting procedure no 'write-in's votes are allowed, the number of candidates m is fixed. A voter may cast any vote including a blank or an invalid votes.

We denote by M a strict upper bound on the number of votes any candidate can receive. In case when each voter has only one vote, M is a strict upper bound on the number of voters n.

We assume that the message space is $\mathbb{Z}_n$ for a suitable $n > M^m$, the l_R-bit randomizer size is $\mathbb{Z}$. For NIZK proofs we use a cryptographic hash-function that outputs an l_e-bit number e. In case of SHA-256 l_e = 256. A security parameter l_s

is such that for any value a, its sum with a random $|a| + l_s$-bit number $a + r_a$ and this random number r_a are indistinguishable. Article [1] suggests to use $l_s = 80$, since this value is large enough to ignore the off chance that $|a + r| > |a| + l_s$.

8.4 Syntax

An e-voting system Π is a quintuple of algorithms and protocols $\langle$ **Setup,Register, Cast,Tally, Result,Verify** $\rangle$ specified as follows:

(1) **Setup**: The algorithm **Setup(**$1^\lambda, \mathcal{P}, \mathcal{V}, \mathcal{U}$**)** is executed by the EA and T. During the setup phase EA generates Π's public parameters $PubEA$ (which include $P, V, U, pkey, snPkey, params$), where $params$ are public election data, $pkey$ is the public key for encrypting votes and $snPkey$ is the public key that is used during the registration process for obtaining a blind signature. EA posts $PubEA$ the on BB . Also, EA the voters' secrets $s_1, \ldots, s_n$, that includes a unique salt seed $seed_i$ for AES encryption, a voter's login $login_i$ and password $password_i$ that are used for signing in. EA distributes secrets among all voters. At the same time, T obtains secret keys $skey$ and $snSkey$ for decrypting ballots and signing respectively and generates pre-election BB data $PubT$ and posts it on BB. We define $Pub = \langle PubEA, PubT \rangle$

(2) **Register**: The algorithm **Register(**$login_i, password_i, id_i$**)** is executed by each voter. A voter V_i randomly chooses a l_b-bit string x_i as his alias and a blinding factor z_i. He uses x_i, z_i and $snPkey$ to construct the value $p_i = Hash(x_i)Enc(z_i; snPkey)$. The voter sends value p_i to the EA along with his $login_i$, $password_i$ and proofs of identity id_i.

(3) **Cast**: The interactive protocol **Cast(**$vote_i, x_i$**)** is executed between tree parties, the voter V_l, the BBand the VC. During this interaction, the voter uses VSD to cast an encrypted ballot $b_i = \langle C, \alpha_i, \pi_i, hash_i, \rangle$, where $C = Enc(vote_i; r)$, $hash_i$ - HASH of entire ballot, α_i and π_i are non-interactive zero-knowledge proofs of eligibility and vote correctness respectively. Upon successful termination, VC posts ballot b_i to BB. The voter V_l receives nothing.

(4) **Tally**: The interactive protocol **Tally(**Pub**)** can be executed by anyone due to a homomorphic property of the encryption scheme. The output is an encrypted sum of all valid ballots τ or $\perp$ in case all entries are invalid.

(5) **Result**: The algorithm **Result**($\tau, sKey$) performs the decryption and outputs the result R_τ of the election and proofs of correct decryption π_τ or returns $\perp$in case such result is undefined.

(6) **Verify**: The algorithm **Verify**(b_i) outputs 1 if ballot b_i is valid and 0 otherwise.

8.5 Building blocks

8.5.1 Blind signature

In cryptography digital signature allows one to 'sign' massage in such a way that everyone can verify the validity of authentic signatures, but no one can forge (to produce a *new* signature) signatures of new messages. A variation on basic digital signatures, known as blind digital signatures, include the additional requirement that a signer can 'sign' a message without knowing what the document contains. The concept of blind signing was introduced by David Chaum in [2].

The global idea is to form a specially constructed message that hides a secret, obtain a signature for this message and then construct a valid signature for the secret. Suppose Alice wants Bob to sign a value x with his private key $skey = (d, N)$ without learning x. To do so, Alice picks random blinding factor z such that $gcd(z, N) = 1$ and calculates $p = xz^e \bmod N$. She sends value p to Bob. Bob signs $p^d \bmod N$ and sends it back to Alice. Alice calculates $p^d \bmod Nz^{-1} = x^d \bmod N$ which is a signature for value x. Bob in not able to determine the secret x on his own due to multiplication on unknown blinding factor. Blind signature can not be broken even with a help of a quantum computer.

Public input: $pkey = (e, N)$ *Private input*: $skey = (d, N)$ **Argument to sign:** $p = xz^e \bmod N$ **Constructing signature**: $p^d \bmod Nz^{-1} = x^d zz^{-1} \bmod N = x^d \bmod N$

Security:
Security of the blind signature scheme relies on an information assumption and does not rely on a computational one, so it achieves information-theoretic security

considered to be cryptanalytically unbreakable. That means that blind signature cannot be broken even if an adversary has unlimited computing power because the adversary simply does not have enough information to break the encryption.

Also, Chaum's blind signature satisfies the blindness property [2] and the non-forgeability [8], [9] of additional signatures, the former means that a signer can not link the blinded message he signs and the original one except with negligible probability, and letter means that after getting l signatures, it is infeasible to compute the $l + 1$ signature.

Informally, unlinkability or blindness can be proven as follows. For any message m there is a unique set of values $r_1, r_2, \ldots, r_n$ that produces a set of blinded messages $m'_1, m'_2, \ldots, m'_n$, with $mi_i \equiv mr_i^e$. No set of values $r_1, r_2, \ldots, r_n$ is any more probable than any other, hence the signer gets no information whether m corresponds to m'_1 or m'_2 or whether it was one of the values signed at all.

Blind signature schemes can be implemented using a common public key signing schemes. One of the simplest blind signature schemes is based on RSA signing.

8.5.2 Homomorphic Integer Commitment and Homomorphic Cryptosystem

There are only a few homomorphic integer commitment schemes [3], [4], [5] and they are quite similar in structure. For the blind signature scheme we offer the following variant. We choose a modulus n as a product of two safe primes and random generators $g_1, \ldots, g_k, h$ of QR_n. In order to commit to integers $m_1, m_2, \ldots, m_k$ using randomness r, we compute $c = com(m_1, m_2, \ldots, m_k; r) = g_1^{m_1} g_2^{m_2} \ldots g_k^{m_k} h^r$. The commitment is statistically hiding if the randomness choice is $r \leftarrow \{0,1\}^{l_r}$.

Root extracting property: When proving soundness and knowledge in our protocols we need a root extraction property, which basically says that if a ciphertext raised to a nontrivial exponent encrypts 1, then the ciphertext itself encrypts 1 [6]. For the homomorphic encryption generalisation of this property can be formalized as follows [1]:

Root extracting property:
If there is a ciphertext C and $e \neq 0$ so $|e| < l_e$ and $C^e = E(M; R)$, then it must be possible to find μ, ρ so $M = e\mu$,$R = e\rho$ and $C = E(\mu; \rho)$.

ElGamal encryption scheme is an asymmetric key encryption algorithm for public-key cryptography with an implementation of Diffie-Hellman key distribution system that was introduced by Taher Elgamal in 1985 [7]. The security of the scheme relies on the difficulty of computing the discrete logarithm over finite fields. ElGamal encryption is semantically secure, has the root extraction property and admit threshold decryption [6].

8.5.3 Proving signature knowledge

Non interactive zero knowledge (NIZK) proof of signature knowledge a method by which one party (the prover) can successfully convince another party (the verifier) that the prover knows something without conveying any information apart from that fact.

Suppose we have a signature σ for a hash value over message x $HASH(x)$. Now we want to convince a verifier that we know the signature without revealing value σ. We choose a random variable A and encrypt it using the public key $pkey = (e, N)$. Then we compute a product S of the encrypted value and our signature raised in power of a challenge c. In the non-interactive proofs the challenge c typically equal to the result of a hash function over the concatenation of all publicly known variables and arguments that is used for proving the statement. The NIZK argument is the challenge c and the product S.

Non-Interactive Zero-Knowledge Argument for proving signature knowledge:

skey: (d,N)
pkey: (e,N)

We prove the signature knowledge by producing:
$NIZK[(x, \sigma) : \sigma = x^d \bmod N]$

Argument:
$\sigma = HASH(x)^d \bmod N$
Choose random $A \in_R Z_N$
$R = A^e \bmod N$
$c = HASH(x||R)$
$S = A\sigma^c \bmod N$
The argument is: (c, S)

Verify:
$\hat{R} = \frac{S^e}{HASH(x)^c} \bmod N$
$\hat{c} = HASH(x||\hat{R})$
Verify $\hat{c} = c$

Indeed if $\hat{c} = HASH(x||\frac{S^e}{HASH(x)^c} \bmod N) = HASH(x||A^e \bmod N) = HASH(x||R) = c$ then proofs of signature knowledge are valid.

8.5.4 Proving that a ciphertext encrypts 0 or 1

Each voter V_i can vote for as many candidates from the set of all candidates as he likes. The voter specifies his choice by setting $a_j = 1$ if he wishes to vote for candidate j and $a_j = 0$ if he does not. The plaintext vote is $VOTE_i = \sum_{j=0}^{m-1} a_j M^j$. The voter encrypts this to get a ciphertext $C = E(\sum_{j=0}^{m-1} a_j M^j; R)$. He now needs to prove that indeed the plaintext is on the right form so $a_j = 0 \vee 1$ for all $j \in [0, m-1]$.

We commit to values $a_0, \ldots, a_{m-1}$. In order to prove that all hidden $a_j \in \{0, 1\}$ we use the fact that $x^2 \geq x$ for any integer and equality holds only for $x = 0$ or $x = 1$. This means that if we want to prove that all a_j's belong to $\{0, 1\}$ we need to convince verifier that $\Delta = \sum_{j=0}^{m-1}(a_j^2 - a_j) = 0$.

Using hide our variables a_j and Δ using the standard techniques $\boxed{a_j} = ea_j + r_{a_j}$ and $\boxed{\Delta} = e\Delta + r_\Delta$. In the verification, we construct the same value $\boxed{\Delta} = \sum_{j=0}^{m-1}(\boxed{a_j}^2 - e\boxed{a_j})$. The left side of this equation is a degree 1 polynomial in e while the right side is a degree 2 polynomial in e. With overwhelming probability over e this implies that the value we committed to $\Delta = 0$, which is exactly what we needed to prove.

The rest of the NIZK argument are a proof of knowledge of the plaintext V and a proof that this plaintext has been properly constructed with values a_j's that we committed to.

Non-Interactive Zero-Knowledge Argument for Correctness of the encrypted vote:

We prove correctness of the vote by producing:
$NIZK[(v, \rho, a_0, \ldots, a_{m-1}) : C = Encr(v; \rho)$ and $v = \sum_{j=0}^{m-1} a_j M^j$
and $\sum_{j=0}^{m-1}(a_j^2 - a_j) = 0]$

Argument:
Choose $r_{a_0}, \ldots r_{a_{m-1}} \leftarrow \{0,1\}^{1+l_s+l_e}$ and let $\Delta = \sum_{j=0}^{m-1}(2a_j - 1)r_{a_j}$
Choose $r \leftarrow \{0,1\}^{l_r}$ and set $c = com(a_0, \ldots, a_{m-1}, \delta; r)$
Choose $r_r \leftarrow \{0,1\}^{l_r+l_s+l_e}$ and set $c_r = com(r_{a_0}, \ldots, r_{a_{m-1}}, \sum_{j=0}^{m-1} r_{a_j}^2; r_r)$
$R_v = \sum j = 0^{m-1} r_{a_j} M^j$, choose $R_R \leftarrow \{0,1\}^{l_R+l_e+l_s}$ and set $C_R = Encr(R_V; R_R)$
Compute a challenge $e \leftarrow hash(C, C_R, c, c_r)$
Set $\boxed{R} = eR + R_R$. Set $\boxed{a_j} = ea_j + r_{a_j}$ and $\boxed{r} = er + r_r$
The argument is: $C_R, c, c_r, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{m-1}}, \boxed{r}$

Verification:
Compute e as above.
Define $\boxed{V} = \sum_{j=0}^{m-1} \boxed{a_j} M^j$ and
$\boxed{\Delta} = \sum_{j=0}^{m-1}(\boxed{a_j}^2 - e\boxed{a_j})$.
Verify $C^e C_R = Encr(\boxed{V}; \boxed{R})$ and $c^e c_r = com(\boxed{a_0}, \ldots, \boxed{a_{m-1}}, \boxed{\Delta}; \boxed{r})$

8.6 System Design

Here is the description of a blind signature e-voting system for r-out-of-m elections, where $0 \leq r \leq m$.
The Blind signature scheme:
Setup ($1^\lambda, \mathcal{P}, \mathcal{V}, \mathcal{U}$).

Let $(GenBL, EncrB, SignBL)$ be the PPT algorithms that constitutive the RSA blind signature scheme, PRG_{str} - a function for pseudo random string generation, PRG_{prime} - a function for pseudo-random prime number generation and $(Gen, Encr, Decr)$ - PPT algorithms for the ELGamal encryption scheme. The EA runs $GenBL(Param, 1^\lambda)$ to generate the blind signature scheme keys (bsk, bpk) and $Gen(Param, 1^\lambda)$ to generate ELGamal keys (sk, pk). Public keys bpk, pk and functions $EncrB, Encr, PRG_{str}, PRG_{prime}$ are posted on the BB.
Then, for every voter V_l, where $l \in [n]$, EA runs $PRG_{str}(1^\lambda)$ function to generate random string $seed_l$.

Registration

Let $(GenAES, EncrAES, DecrAES)$ be the publicly known PPT algorithms that implements AES encryption scheme and $Hash$ - hash algorithm.

Every voter V_l completes the following procedure:
– uses the published on the BB function $PRG_{str}(1^\lambda)$ to generate his alias x_l and $PRG_{prime}(bpk)$ to generate a blinding factor z_l.
– calculates value $p_l = Hash(x_l)EncrB(z_l)$.
– chooses his secret password $password_l$ and runs $GenAES(password_l, s_l)$ to generate his key for symmetric encryption key_l, where $s_l = PRG_{str}(seed_l)$.
– encrypts x_l and z_l by running $EncrAES(key_l, x_l)$ and $EncrAES(key_l, z_l)$ respectively to get encrypted values $\hat{x}_l, \hat{z}_l$
– sends $p_l, \hat{x}_l, \hat{z}_l$ to the EA

EA:
Upon receiving $p_l, \hat{x}_l, \hat{z}_l$ from a voter, EA posts all information to the BB.
When registration is closed, EA runs $SignBL(bsk, p_l)$ for every entry in the BB and post the result in the corresponding line as p_l^{sign}.

Cast:

Let $e_l = (e_{1_l}, e_{2_l}, \ldots, e_{m_l})$ be the characteristic vector corresponding to the voter's selection, where $e_{j_l} = 1$ if the option opt_j is selected by the voter V_l.
Every voter V_l completes the following procedure:
– gets all information from the BB - $\{p_l, \hat{x}_l, \hat{z}_l\}$.
– finds his entry and decrypts x_l and z_l by running $DecrAES(key_l, \hat{x}_l)$ and $DecrAES(key_l, \hat{z}_l)$ respectively, where $key_l = GenAES(password_l, s_l)$ and $s_l = PRG_{str}(seed_l)$
– computes σ_l - signature for x_l by calculating $p_l^{signed} z_l^{-1}$
– computes Π_l – NIZK proof of signature knowledge .
– chooses his vote-option e_{j_l} and writes the corresponding characteristic vector e_l
– for $j \in [m]$ compute $c_{j_l} = Encr(pk, e_{j_l})$
– computes NIZK proofs π_{j_l} that each c_{j_l} is an encryption of 1 or 0.
– sends $b_l = (x_l, \sigma_l, \Pi_l, c_l, \pi_l)$ to the EA.
– keeps x_l as receipt.
– If EA accept b_l, protocol terminates successfully.
*Optional: every voter can export the randomness to check that the ballot was cast as intended.

Upon receiving $(x_l, \sigma_l, \Pi_l, c_l, \pi_l)$ from a voter, EA checks NIZK proofs and, if it's valid, accepts the ballot and posts all information to the BB.

Tally:
After election is closed, EA computes $\mathcal{C}$ – the sum of all c_l and runs $Decr(sk, \mathcal{C})$ to decrypt Tally τ. EA posts the τ along with the proof of tally correctness $Proof$.

Result:
result is straightforward.

Verify:
The algorithm returns 1 only if the following checks are true:
– exported randomness is correct or voter choose not to check.
– there is ballot with x_l
– all Π_l, π_l are valid
– number of ballots less or equal to the number of registered voters
– $Proof$ is correct
– sum of all scores at τ are less than or equal to ballot numbers

8.7 E2E Verifiability

In the E2E verifiability proof, only BB is assumed to be honest. The rest components of the election server are controlled by the adversary and its is allowed to change the content on the BB arbitrarily before the **Result** protocol starts. However, we can assume that validity of all NIZK proofs on the BB can be checked by anyone as well as the result of the **Tally** protocol execution, since $Tally$ is computed homomorphically and this computation doesn't require any secret information. To prove that the blind signature scheme is E2E verifiable, we first construct a vote extractor $\mathcal{E}$:

$\mathcal{E}$ has input τ and the set of receipts $\{x_l\}_{V_l \in \tilde{\mathcal{V}}}$ where $\tilde{\mathcal{V}}$ is the set of the honest voters that voted successfully. If $Result(\tau) = \perp$ (i.e., the transcript is not meaningful), then $\mathcal{E}$ outputs $\perp$. Otherwise, $\mathcal{E}$ (arbitrarily) arranges the voters in $\mathcal{V} \backslash \tilde{\mathcal{V}}$ and the tags not included in $\{x_l\}_{V_l \in \tilde{\mathcal{V}}}$ as $\langle V_l^{\mathcal{E}} \rangle_{l \in [n-|\tilde{\mathcal{V}}|]}$ and $\langle tag_l^{\mathcal{E}} \rangle_{l \in [n-|\tilde{\mathcal{V}}|]}$ respectively. Next, for every $l \in [n - |\tilde{\mathcal{V}}|]$:

(1) $\mathcal{E}$ finds at the BB entry with $x_l = tag_l^{\mathcal{E}}$ and brute-force the corresponding ELGamal cipher to open the selected candidate $\mathcal{P}_l^{\mathcal{E}}$. If $\mathcal{P}_l^{\mathcal{E}}$ is the valid candidate's selection, then $\mathcal{E}$ sets $\mathcal{U}_l^{\mathcal{E}} = \{\mathcal{P}_l^{\mathcal{E}}\}$. Otherwise it inputs $\perp$.

Finally $\mathcal{E}$ outputs $\langle \mathcal{U}_l^{\mathcal{E}} \rangle_{V_l \in \mathcal{V} \backslash \tilde{\mathcal{V}}}$ Finally $\mathcal{E}$ outputs $\langle \mathcal{U}_l^{\mathcal{E}} \rangle_{V_l \in \mathcal{V} \backslash \tilde{\mathcal{V}}}$

According to the definition of E2E verifiability, an adversary $\mathcal{A}$ wins the game $G_{\text{e2e-ver}}^{\mathcal{A},\mathcal{E},d,\theta}(1^\lambda, n, m, t)$ and breaks E2E verifiability if it allows at least θ honest voters to cast their votes successfully and achieves tally deviation d.

All NIZK proofs are perfectly sound and the **Tally** protocol is completely transparent and can be repeated and checked by anyone.
Modification attack - we need a hash of the whole ballot and ballot id as receipt to prevent malicious EA from cheating. $\mathcal{A}$ may attempt to decrypt a single vote, learn x, create a signature and compute a perfectly valid vote for another candidate.

However EA can use abstain voters to create additional ballots and cause deviation. It can use $y_{reg} - y_{voted} - 1$ additional votes. Abstain voter can not prove anything since he doesn't know whether he is the only one who abstains or not. However, if every registered voter is assigned a specific entry in BB he can detect

that someone (EA) voted on his behalf. On the other hand, assigning every voter ballot id makes this system linkable. Tradeoff: either EA should be able to link an individual voter with his ballot or EA can vote on behalf of abstain voters and no-one would detect it as long as number of additional ballots strictly less than the number of abstain voters.

References

[1] J. Groth, "Non-interactive zero-knowledge arguments for voting," *Applied Cryptography and Network Security*, pp. 467–482, 2005.

[2] D. Chaum, "Blind Signatures for Untraceable Payments," 1982.

[3] F. Eiichiro and T. Okamoto, "Statistical zero knowledge protocols to prove modular polynomial relations," *In proceedings of CRYPTO '97, LNCS series*, vol. 1294, pp. 16–30, 1997.

[4] I. Damgard, E. Fujisaki, and I. Damg, "An Integer Commitment Scheme based on Groups with Hidden Order," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2501, pp. 125–142, 2002.

[5] J. Groth, "Cryptography in Subgroups of Z_N*," *Theory of Cryptography*, vol. 3378, pp. 50–65, 2005.

[6] J. Groth, "A verifiable secret shuffle of homomorphic encryptions," *Journal of Cryptology*, vol. 23, no. 4, pp. 546–579, 2010.

[7] T. Elgamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[8] D. Pointcheval and J. Stern, "Provably Secure Blind Signature Schemes," *Advances in CryptologyASIACRYPT'96*, vol. 96, pp. 252–265, 1996.

[9] R. E. Abstract, A. Juels, M. Luby, and R. Ostrovsky, "Security of Blind Digital Signatures," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1294, no. 1, pp. 150–164, 1997.

[10] R. L. Rivest, S. Goldwasser, S. Micali, and R. L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks *," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988.

[11] A. Kiayias, T. Zacharias, and B. Zhang, "End-to-end verifiable elections in the standard model," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9057, pp. 468–498, 2015.

[12] A. Kiayias and T. Zacharias, "DEMOS-2 : Scalable E2E Verifiable Elections without Random Oracles," *Ccs '15*, vol. 2015-Octob, pp. 352–363, 2015.

[13] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "SoK: A comprehensive analysis of game-based ballot privacy definitions," *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2015-July, pp. 499–516, 2015.

[14] D. Bernhard, V. Cortier, O. Pereira, B. Smyth, and B. Warinschi, "Adapting helios for provable ballot privacy," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6879 LNCS, pp. 335–354, 2011.