Exploring boundaries of privacy in E2E verifiable e-voting systems

by

Tamara Finogina

Submitted to the Department of Computational Science and Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

SKOLKOVO INSTITUTE OF SCIENCE AND TECHNOLOGY

May 2017

Author . . . .
Department of Computational Science and Engineering
May 11, 2017

Certified by . . . .
Stamatis Lefkimmiatis
Assistant Professor
Thesis Supervisor

Exploring boundaries of privacy in E2E verifiable e-voting systems

by

Tamara Finogina

Submitted to the Skolkovo Institute of Science and Technology
on May 11, 2017

Abstract

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.

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 limits of privacy that is possible to achieve in any E2E verifiable e-voting system.

In this work, we perform a thorough and formal study on 'locating' the critical contradiction point in the voter privacy-E2E verifiability tradeoff. As part of this analysis, we introduce a strong privacy definition where voters are corrupted but an adversary is still unable to break privacy, denoted as strict privacy. We formally define strict voter privacy via a Voter Privacy game that is played between an adversary A and a challenger C . According to the game rules, an adversary is allowed to define the election parameters, corrupt a number of entities, and act on behalf of all voters. As for we apply the E2E verifiability definition given by Kiayias et al. [30], according to which even when all election administrators are corrupted, they can not manipulate the results without a high detection probability.

Under this framework, we prove that strict privacy is the weakest level of privacy that contradicts end-to-end verifiability. Namely, any meaningful relaxation of the strict privacy definition, leads to a notion of privacy that is feasible by some E2E verifiable e-voting system.

Also, we have developed and implemented an e-voting scheme that is based on blind signature scheme approach for illustrating privacy limitations.

Thesis Supervisor: Stamatis Lefkimmiatis
Title: Assistant Professor

Acknowledgments

I would like to thank all the people who contributed in some way to the work described in this thesis. Foremost, I would like to express my sincere gratitude to Dr. Thomas Zacharias for the continuous support of my MS study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

My thanks appreciations also goes to Prof. Aggelos Kiayias for his unending support, advises and effort to make this study possible.

I offer my sincerest gratitude to my supervisor, Prof. Stamatios Lefkimmiatis, who has supported me throughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way.

My sincere thanks also goes to Jordi Puiggalí, Sandra Guasch and Nuria Costa for offering me the summer internship at Scytl and for their valuable comments and support. I would also like to thank all the members of Scytl R&D team for their continued support and encouragement.

Furthermore I would like to thank Prof. Panagiotis Karras for introducing me to the topic as well for the support on the way.

I would like to thank Skoltech and its stuff for all opportunities that you gave me.

Last but not the least, I would like to thank my loved ones, who have supported me throughout entire process. I will be grateful forever for your love.

Thanks for all your encouragement!

Contents

Chapter 1

Introduction

Electronic voting (e-voting) is a term used to describe the act of voting using electronic systems. Generally, the term e-voting is applicable for two different types of electronic voting scenarios: 1) 'Remote e-voting' voting over the Internet via personal gadgets (laptops, smartphones, etc.) at any place outside the polling station and 2) 'Polling place e-voting' – voting inside a polling station or similar premises controlled by electoral staff. In this work, both meanings are used.

Electronic voting has been in the centre of researchers attention for over the last twenty years. Up to now, many e-voting schemes with quite strong security guarantee have been proposed. Perhaps the most well known and studied system is Helios designed by Ben Adida [2]. Helios is an open-source purely cryptographic voting protocol which does not rely on paper. Other examples of purely electronic systems are Demos-2 [30] and Civitas [16]. Another subclass of e-voting systems is so-called hybrid systems where paper ballots are used for computing tally or ensuring the integrity of an election. Demos [31], ThreeBallot [40], Prêt-à-Voter [41] and Scantegrity [14] are examples of hybrid systems.

Nowadays, some countries allow their citizens who are living or staying abroad to vote remotely. However, only a few countries allow external voters to cast their votes electronically. For the last sixty years e-voting (including remote e-voting) has been conducted at least once only in the following countries Australia, Belgium, Brazil, Canada, Estonia, France, Germany, India, Italy, the Netherlands (Rijnland Internet Election System), Norway, Peru, Romania, Switzerland, the UK, Venezuela, and the Philippines (according to Wikipedia [44]).

Benefits of using electronic voting are significant:

1. Voting is easier and more convenient;
2. E-voting system can be completely auditable. Therefore elections are completely transparent;
3. Results are announced faster;
4. Increased engagement and turnout;
5. Increased accessibility;
6. Voting is provably secure.

However, all of the potential benefits are moot if we can not trust the election results. Some argue that electronic voting is not secure because of issues with the technology, vast possibilities of fraud, and protection of voters privacy. To eliminate the security risks a number of security requirements for e-voting systems were developed: ensuring one vote per voter, voters eligibility, maintaining voter anonymity, the accuracy of tallying and prevention of fraud.

Among the security properties that have been identified for e-voting, there are two highly desirable properties that are considered to be crucial and has been investigated extensively: E2E Verifiability and Voter Privacy.

E2E Verifiability means that it is possible to verify the correctness of the election outcome based on feedbacks from voters and examination of the public election transcript. Informally the E2E verifiable property means that any voter can detect that the election outcome has been manipulated. Formally, E2E verifiability is defined as the ability of a voter to verify that: 1) his vote was properly cast, 2) recorded as cast and 3) tallied into the election result as recorded.

Voter Privacy can be described as follows: e-voting system should not reveal how a particular voter voted. Voter privacy can be divided into tree levels of different strength (in increasing order):

1. Ballot privacy: A set of ballots would not reveal voters' choices to anyone [8].

2. Receipt-freeness: An honest voter cannot prove to an adversary that he voted in a certain way [32].

3. Coercion resistance: A corrupted voter cannot prove to an adversary that he voted in a certain way [19].

It has been observed that voter privacy and E2E verifiability requirements inherently contradict each other at some point. In this work, we perform a thorough and formal study on privacy in E2E verifiable e-voting systems to analyse those restrictions and define the privacy limits. We suggest a stronger privacy definition, that does not impose restrictions on an adversarial behaviour. Also, we define a notion of strict privacy and prove in simulation-based settings that it contradicts E2E Verifiability. At the final chapter of this work, we present a blind signature scheme, that captures the idea of voting anonymously, and prove that the whole class of such systems is not E2E Verifiable.

1.1 Preliminaries

Through this paper, security parameter is denoted as λ. The notion $negl(\lambda)$ is used 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.

Let Π be an e-voting system and $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of m candidates. Voters $\mathcal{V} = \{V_1, \ldots, V_n\}$ use Π to vote for some allowed subset of candidates selections from the collection of allowed selections $\mathcal{U}$ (including 'blank' option as well).

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 .

1. BB is completely passive and provides storage for the election transcript for the verification purpose. It is writeable by the VC, T and EA and readable by anyone.
2. Voters submit their votes by starting the **Cast** protocol with VSD and providing their credentials and preferences as input. The main restriction is that voters 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.

In many election systems, the EA and T are implemented by more than a single authority. However, we consider an entity to be malicious as a whole, therefore, for simplicity in the syntax, we assume that both EA and T are single entities.

1.2 Syntax and Correctness

An e-voting system Π is a quintuple of algorithms and protocols **(Setup, Cast, Tally, Result, Verify)**, that takes voters' preferences as an input and aims to return a tally, the protocols 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 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.

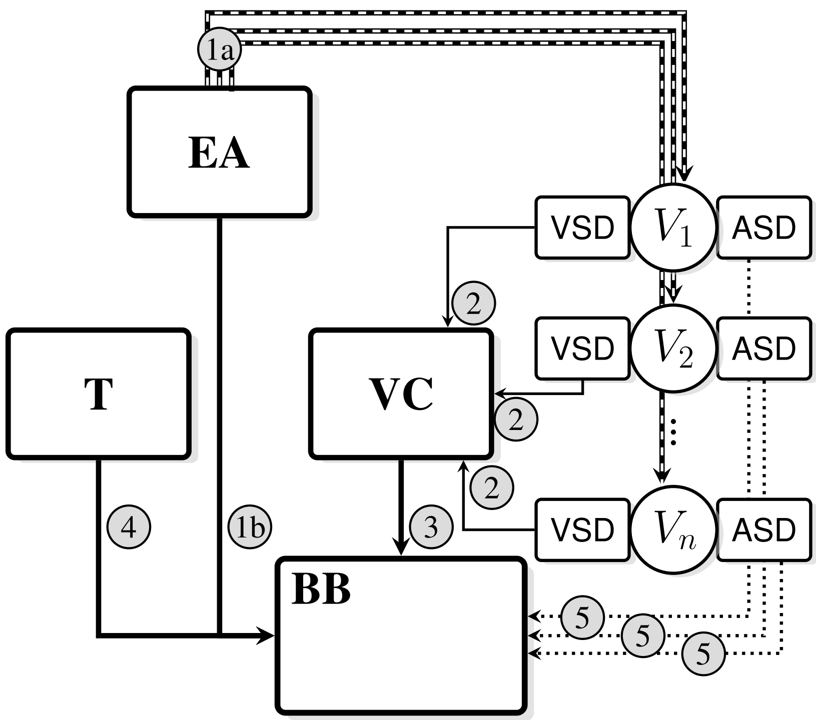


Figure 1-1: The interaction among the entities in an e-voting execution. The dotted lines denote read-only access to the BB. The dashed arrows denote channels for voters' private inputs distribution. Annotation: (1a): distribution of voter's private inputs; (1b): posting pre-election BB data; (2): vote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

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 $\perp$ if result is undefined.

4. The algorithm **Verify** on input α, τ outputs a bit that determine whether the verification was successful or not. α is a voter receipt obtained after the **Cast** protocol execution.

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

Definition 1.2.1 (Correctness of e-voting system by Kiayias et al. [30]). It is said 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 $U_1, ..., U_n$.

1.3 Election process

Any electronic voting procedure can be split into three general stages: Pre-election, Election and Post-election. For some specific e-voting schemes, the first or the last stage can be omitted.

1. Pre-election: EA generates public pre-election data and posts it on the BB. Also, EA creates and distributes envelopes 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. Election: this stage typically is split into two parts:

(a) Registration: A voter and EA engaged in an interaction during which the voter proofs to the EA his identity and obtains credentials for vote casting. In some e-voting systems, the registration part is omitted, and voters receive their credentials in envelopes via some secure channel.

(b) Vote casting: A 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.

3. Post-election: After the election is closed, the election result is computed and announced by T. If verification is supported, anyone may check the validity of the election procedure.

1.4 E2E Verifiability

E2E verifiability is a very strong level of security that allows voters to detect that a malicious e-voting system tries to misrepresent the election outcome. Three aspects of verifiability are usually distinguished:

1. *Individual verifiability*: a voter can check that his ballot is counted correctly [15].

2. *Universal verifiability*: anyone can check that the election outcome is obtained from the ballots published on the BB [42].

3. *End-to-end verifiability*: a voter can check that his vote was cast-as-intended and recorded-as-cast, also anyone can check that the ballots were tallied-as-recorded [37], [13].

In this work, we use the E2E-verifiability definition by Kiayias et al. [31], [30], the definition is listed in Figure 1-2. According to the definition, an adversary $\mathcal{A}$ can control all VSD's, EA, VC and some fraction of voters, however, it still can not manipulate the results without a high detection probability. The entities involved are BB, VSD and EA that can be split on EA, that involved on pre-election stage only, and T that computes the results. The algorithm **Cast** is run interactively between tree parties the BB, the voter V_l

and T that uses VSD with the following inputs: public parameters Pub, voter's secret s_l and voter's choice $\mathcal{U}_l$. Upon successful termination, V_l obtains a receipt α_l. The algorithm **Verify**(τ, α_l) outputs a bit, based on voter's receipt α_l and the public transcript τ. The algorithm **Result**(τ) given the public transcript τ outputs the result for the election or $\perp$ if the result is undefined.

We prefer the definition by Kiayias et al over the one given by Küsters et al [33], because it is given in game-based settings. Moreover, this definition of E2E Verifiability does not require any additional assumptions, except for the existence of the BB, therefore it is achievable in the standard model. Furthermore, there is an ideal functionality that captures the essential aspects of the E2E Verifiability [3].

E2E Verifiability Game $G^{\mathcal{A},\mathcal{E},d,\theta}_{E2E-Ver}(1^\lambda, m, n)$:

1. $\mathcal{A}$ chooses a list of candidates $\mathcal{P} = \{P_1, \ldots, P_m\}$, a set of voters $\mathcal{V} = \{V_1, \ldots, V_n\}$ and the set of allowed candidate selections $\mathcal{U}$. It provides $\mathcal{C}$ with the sets $\mathcal{P}$,$\mathcal{V}$,$\mathcal{U}$ along with information Pub and voter credentials $\{s_{l\in[n]}\}$. Throughout the game, $\mathcal{C}$ plays the role of the BB.
2. The adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ engages in an interaction where $\mathcal{A}$ schedules the **Cast** protocols of all voters. For each voter V_l $\mathcal{A}$ can either completely control the voter or allow $\mathcal{C}$ to operate on their behalf, in which case $\mathcal{A}$ provides a candidate selection $\mathcal{U}_l$ to $\mathcal{C}$. Then, $\mathcal{C}$ engages with the adversary $\mathcal{A}$ in the **Cast** protocol so that $\mathcal{A}$ plays the role of EA. Provided the protocol terminates successfully, $\mathcal{C}$ obtains the receipt α_l on behalf of V_l.
   Let $\tilde{\mathcal{V}}$ be the set of honest voters (i.e., those controlled by $\mathcal{C}$) that terminated successfully.
3. Finally, $\mathcal{A}$ posts the election transcript τ to the BB.

The game returns a bit which is 1 if and only if the following conditions hold true:
(i) $|\tilde{\mathcal{V}}| \geq \theta$, (i.e., at least θ honest voters terminated).
(ii). $\forall l \in [n]$: if $V_l \in \tilde{\mathcal{V}}$, then **Verify**$(\tau, \alpha_l)$=1 (i.e., the voters in $\tilde{\mathcal{V}}$ verify their ballot successfully).
and either one of the following two conditions:
(iii-a). If $\perp \neq \rangle \mathcal{U}_l \rangle_{V_l \in \mathcal{V} \setminus \tilde{\mathcal{V}}} \leftarrow \mathcal{E}(\tau, \{\alpha_l\}_{V_l \in \tilde{\mathcal{V}}})$,
then d_1(**Result**(τ),$f(\langle U_1, \ldots, U_n \rangle)) \geq d$.
(iii-b). $\perp \leftarrow \mathcal{E}(\tau, \{\alpha_l\}_{V_l \in \tilde{\mathcal{V}}})$.

Figure 1-2: E2E-verifiability by Kiayias et al.

1.5 The Universal Composability Framework

The Universal Composability (UC) is a framework for representing cryptographic protocols and analysing their security [10]. UC provides very strong security guarantees and allows to specify security requirements in a unified and systematic way. There are just tree entities involved: a protocol, an adversary and an environment that captures everything else that goes beyond the protocol execution. The best explanation what does UC mean is given by Jens Groth: "In the UC framework, an execution of a multi-party computation protocol is compared to an execution where a trusted ideal functionality handles the data and produces the output. A protocol is said to be secure if an adversary operating in a real-life model can be simulated in the ideal process model with the ideal functionality. In the case of voting, the ideal functionality takes as input the votes and outputs the result of the election. This ideal functionality corresponds to the old method of voters marking their choice on paper and putting the ballot in a box, which is opened once the election is over." [24].

Originally UC considers the execution of an unbounded number of concurrent protocols in the arbitrary environment, controlled by an adversary. Even though UC seems to be the most realistic level of security, it cannot be achieved in general without some trusted setup assumptions [35]. The stand-alone setting on the other hand only allows the execution of a single instance of the protocol at a time and guarantees the security under sequential composition, namely if a protocol is secure in the stand-along model it maintains its security in sequential runs, where each execution concludes before the next one begins. Sequential composition does not imply security in the concurrent composition, however, it provides reasonably high-security guarantees.

In the stand-alone model, an adversary is restricted from communicating with the environment during the protocol execution. The stand-alone model enables one to design a protocol using calls to an ideal functionality, which is secure by design, and check whether a real protocol properly implements this ideal protocol. This makes security analysis significantly more simple. A protocol is said to be secure if, for all adversaries, there exists a simulator so that real and ideal executions are indistinguishable for and any environment [36].

1.6 Random Oracle

Random Oracle Model was invented as an artificial way to construct formal security proofs for certain types of cryptographic protocols [6]. The model can be described as a black box with a perfectly random function we know nothing about . We can input some new date into the box (ask quires) and receive an uniformly distributed random output. However, if given an already seen input, the box outputs whatever it returned the last time.

Basically, Random Oracle is an perfect hash function i.e. a *random function* with memory. Unfortunately, random functions are impractical – it is just an model, that can not be implemented in the real life. There exist signature and encryption schemes that are secure in the Random Oracle Model, but for which any implementation of the random oracle results in insecure schemes. Therefore for constructing security proofs in Random Oracle model, we pretend that some real implementation of the hash function is perfectly random, even though it is decidedly not random. The problem with this approach is that it is unclear what security guarantees it buy us in the real world. Canetti et al. [11] prove that there are protocols that secure in Random Oracle model, but any implementation of such protocols results in insecure schemes. However, despite everything, security proofs in Random Oracle model are much better than no proofs at all and this model was wildly used for proving security of RSA, ElGamal etc.

Definitions are given in the Standard and Random Oracle models are incomparable. The Standard model implies that security relies on the standard cryptographic assumptions, while in Random Oracle we assume that our implementation of the hash function is perfectly random, even thought it is not.

Chapter 2

Privacy

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. Similarly to [30], 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, except for untrappable channels, 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.

During the election process, a voter uses his credentials to cast a vote for some 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 is 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 plain text, or by trusted VSD and crypto performed inside it.

The approach that we used in this work 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 $\mathcal{C}$. The game 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 honest and (2) VSD is honest.

2.1 Why privacy contradicts E2E Verifiability?

The intuition behind the existence of the 'critical point', where Voter Privacy starts to contradict E2E Verifiability, is the following. Informally, Voter Privacy states that no one should be able to link a particular voter and his vote for a certain period of time, usually quite long. If there is any kind of connection between a voter and his choice, an adversary should not be able to find it in a reasonable time. In the extreme case of Everlasting Voter Privacy, such link, if exists, should be information-theoretically secure, otherwise an adversary would be able to find it and break the privacy. On the other hand, verifiability mans that there is a way to ensure that system works properly. In case of e-voting, verifiability implies that an election outcome is trusted if some assumptions holds. Moreover, public verifiability can be viewed as *likability* of all voters with the set of received and recorded ballots [34]. At the high level, the contradiction is quite obvious: privacy attempts to remove any link at all and verifiability requires the existence of the same link.

2.2 Why do we need another privacy definition?

In this work we are looking for the point where Voter Privacy starts to contradict E2E Verifiability. To find this ’critical point’ we need to define Voter Privacy and E2E Verifiability first.

As for E2E Verifiability, we use the existing definition by Kiayias et al. [30], which states that the outcome of an election can not be misinterpreted even if the whole system is malicious and only some fraction of voters is honest. It is a quite strong approach, that does not rely on any additional assumptions. Moreover, this level of E2E Verifiability can be achieved information theoretically in the Standard Model (example Demos [30]). Details are given in section 1.4.

Most known Voter Privacy or rather Ballot Privacy definitions (see section 2.5 for details) are specific to the Random Oracle Model and therefore is not compatible with the Standard Model as discussed in sections 1.6 and 1.5. The only Voter Privacy definition, we are aware of, applicable to the Standard Model is the Demos Privacy [30] (Details are in section 2.5.2). However, the Demos Privacy game is given only for the case when an adversary commits to two lists of choices that results in the same tally. If, for example, the adversary would use any other lists it loses by default.

The goal of this work is to find the contradiction between Voter Privacy and E2E Verifiability. Therefore, we are looking for such privacy level that allows an adversary $\mathcal{A}$ to win E2E Verifiability game with the probability more than one-half. The E2E Verifiability game states that $\mathcal{A}$ wins only if there is a noticeable (more than a give threshold) tally deviation and honest voters successfully verify their ballots. $\mathcal{A}$ can win the game only if it somehow modifies ballots of honest voters or find a way to justify the wrong tally.

So for the attack against E2E Verifiability to be successful, we need Demos Privacy to be able to handle lists that do not summed to the same result. Therefore we need to modify the existing definition or to construct a new one, that is at least Demos strong but allows an adversary to work with lists that are summed up to the different results.

Also, we have noticed that, from a voter point of view, there are only two ways to ensure privacy: rely on crypto performed inside VSD or trust pre-computed by EA credentials. An honest voter can preserve his privacy only if he is allowed to interact with VSD secretly and/or keep his credentials in secret. In Demos Privacy, an honest voter is allowed to do both. Moreover, the challenger returns to an adversary only one view, which is real or fake depending on the challenger's bit.

Our privacy approach is a bit different, 1) we allow only one hidden from the adversarial eyes interaction and 2) we return both views. The former requirement split our privacy in two scenarios: a) an honest voter can keep his credentials in secret and b) an honest voter can interact with the VSD in secret. The latter grands an adversary more power, since it has access to more information. The additional information provided to an adversary makes our privacy definition stronger, but more complicated.

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

The $\mathcal{C}$'s strategy in the $G^{\mathcal{A},Sim}_{strict,\mathrm{EA},\mathrm{T}}(1^\lambda, n, m)$ game captures the ability of an honest voter to lie about his vote in code-based e-voting schemes. Suppose, an adversary $\mathcal{A}$ makes a voter V to vote for an option $U_{\mathcal{A}}$. However, V disobeys and votes for his intent U_V. If V can fake his credentials and convince $\mathcal{A}$ that he voted for $U_{\mathcal{A}}$, V's privacy is preserved. This approach is similar to *Demos Privacy* [30], however instead of faking voter's internal view

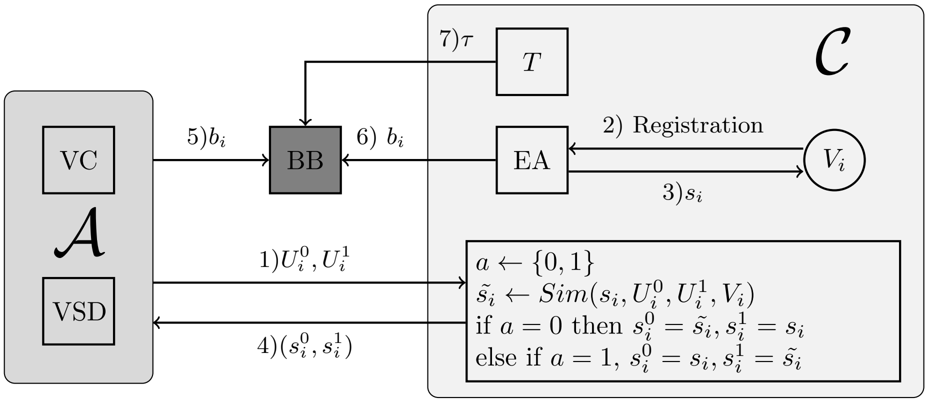


Figure 2-1: $G^{\mathcal{A},Sim}_{t-priv,\mathrm{EA},\mathrm{T}}(1^\lambda, n, m)$

of the **Cast** protocol, we fake credentials in a way that would produce a fake internal view of the **Cast** protocol.

We compare *Demos Privacy* and the *Voter Privacy* defined here as follows. Under *Demos Privacy* framework a voter V provides $\mathcal{A}$ with a fake internal view of the **Cast** protocol and the actual receipt as a proof of obedience but keeps the original credentials in secret. If $\mathcal{A}$ accesses the voter's original credentials, it would immediately detect the lie. Under our *Voter Privacy* definition, a voter V gives $\mathcal{A}$ a fake view, actual receipt and fake credentials. If $\mathcal{A}$ can not distinguish real and fake credentials, it has no way to find out whether V lies or not.

In the game $G^{\mathcal{A},Sim}_{t-priv,\mathrm{EA},\mathrm{T}}(1^\lambda, n, m)$, 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.

1) $\mathcal{A}$ picks and sends two options U_i^0, U_i^1 to the challenger $\mathcal{C}$. The first option U_i^0 is its intent, the other – the option that the challenger would use in order to produce an indistinguishable from the intent's ballot and receipt view*. 2) After sending options, $\mathcal{A}$ schedules the **Registration** protocol with $\mathcal{C}$ on behalf of some voter V_i. 3) $\mathcal{C}$ creates fake credentials $\tilde{s}_i$ and generates real credentials s_i for the voter V_i. 4) $\mathcal{C}$ responds $\mathcal{A}$ with a pair of cre-

dentials s_i^0, s_i^1, where one of the credentials are real and the other were generated using the simulator Sim in a such way, that if $\mathcal{A}$ guesses right and uses the real credentials to cast a vote for U_i^0, the produced ballot and internal view of the **Cast** protocol would be real, otherwise generated ballot would correspond to the option U_i^1 and the returned view would be fake. 5) If $\mathcal{A}$ chooses to post the ballot b_i to BB, 6) $\mathcal{C}$ posts exactly the same ballot** to BB. 7) When $\mathcal{A}$ stops the election, $\mathcal{C}$ posts the tally τ ***.

Remarks:

Denote the list of honest voters for which $\mathcal{A}$ chooses to post produced ballot and uses credentials s_i^0 as $\tilde{\mathcal{V}}^0$ and the similar list but for credentials s_i^1 as $\tilde{\mathcal{V}}^1$.

*If $\mathcal{C}$ succeeds, $\mathcal{A}$ wouldn't be able to say whether it voted for option U_i^0 or U_i^0 and BB would contain both ballots (one for the option U_i^0 ,the other for the option U_i^1) so the tally wouldn't reveal any information. Example: if $\mathcal{A}$ picks real credential then it casts a vote for the option U_i^0 on behalf of a voter V_i, at the same time exactly the same ballot posted by $\mathcal{C}$ would correspond to the fake credentials and the option U_i^1. In this case, $\mathcal{A}$ indeed voted for U_i^0 as it intended. Else if $\mathcal{A}$ picks the fake credentials and votes for the option U_i^0, its ballot corresponds to the real credentials and the option U_i^1. At the same time, exactly the same ballot posted by $\mathcal{C}$ corresponds to the fake credentials and the option U_i^0. In the last case $\mathcal{A}$ actually voted for U_i^1 while thinking it casts vote for U_i^0. If this privacy holds, $\mathcal{A}$ has no idea what option voter V_i voted for.

** $\mathcal{C}$ should post to the BB a ballot for the remaining candidate and since $\mathcal{C}$ does not control VSD, it can only post the identical ballot, but **Tally** it as if it was generated with fake credentials. If $\mathcal{A}$ voted for U_i^0, $\mathcal{C}$ should vote for the U_i^1 and vice versa. Otherwise $\mathcal{A}$ would guess the coin a by simply checking the result of an election. $\mathcal{A}$ is allowed to pick any two options U_i^0, U_i^1 and play as many rounds as it likes. Since lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ not necessarily produce the same result as lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$, the adversary would trivially break privacy. To prevent this, we add challenger's ballots to the BB and compute the combined tally, namely $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1})$.

However, if lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ sums to the same result $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, we remove the challenger's ballots and compute the actual tally.

*** The tally τ is posted only if, for every correctly formed adversarial ballot, there is a corresponding challenger's ballot posted on the BB. $\mathcal{A}$ may not post some ballots, however, if it does, then the corresponding challenger's ballot must be posted as well. Otherwise, $\mathcal{A}$ wins the *Voter Privacy* game by simply checking the announced result. All $\mathcal{A}$'s ballots are tallied based on real credentials, all $\mathcal{C}$'s ones – based on fake credentials. $\mathcal{C}$ removes all its ballots and computes tally for the adversarial ballots if and only if lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed up to the same result i.e. $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$. In this case, the announced tally would be $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1})$ if challenger's coin $a = 0$ or $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$ otherwise.

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

1. $\mathcal{A}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, chooses 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 $a \leftarrow \{0, 1\}$ to define an order according to which real and simulated credentials would be returned to $\mathcal{A}$, and starts the election on behalf of EA.

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 the $\mathcal{A}$ 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 and 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 forwards 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. 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. Denote the list of honest voter for which $\mathcal{A}$ chooses to post produced ballot and uses credentials s_i^0 as $\tilde{\mathcal{V}}^0$ and the similar list but for credentials s_i^1 as $\tilde{\mathcal{V}}^0$. If lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed up to the same result i.e. $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, $\mathcal{C}$ removes all its ballots and executes the **Tally** protocol on the cleared BB with real credentials. Otherwise, $\mathcal{C}$ executes the **Tally** protocol on the BB that contains both adversarial and challenger's ballots and uses real credentials for adversarial ballots and fake ones for its own.

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 $\hat{\mathcal{V}} = \mathcal{V} \backslash \mathcal{V}_{corr}$. 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 such a way that fake credentials satisfy both of the following rules:

1. The real credentials and U_i^0 option should give ballot and receipt, which are identical to ballot and receipt produced for the fake credentials and U_i^1 option.

2. The fake credentials and U_i^0 option should give ballot and receipt, which are identical to ballot and receipt produced for the real credentials and U_i^1 option.

To understand the logic behind the *Voter Privacy* defined above, consider the following toy examples.

Example 1

We have three voters V_1, V_2, V_3 and three possible options op_1, op_2, op_3. Challenger's coin $a = 0$, so, during the game, the fake credentials are always the first credentials in the credentials pair. Suppose $\mathcal{A}$' schedules three vote casting protocols for voters V_1, V_2, V_3:

1)$\boxed{V_1}$: $\mathcal{A}$ chooses $U_1^0 = op_3, U_1^1 = op_1$ and picks the second credentials for generating the ballot b_1 for op_3. Since $\mathcal{A}$ picked the real credentials, the ballot b_1 in the **Tally** process would correctly result in a vote for op_3. Meanwhile, $\mathcal{C}$ posts identical to the ballot b_1 ballot $\tilde{b_1}$ that corresponds to an option op_1.

2)$\boxed{V_2}$: $\mathcal{A}$ chooses $U_2^0 = op_1, U_2^1 = op_2$ and picks the first credentials for generating the ballot b_2 for op_1. Since $\mathcal{A}$ picked the fake credentials, the ballot b_2 in the **Tally** protocol would result in a vote for op_2. Meanwhile, $\mathcal{C}$ posts identical to the ballot b_2 ballot $\tilde{b_2}$ that corresponds to an option op_1.

3) $\boxed{V_3}$: $\mathcal{A}$ chooses $U_3^0 = op_3, U_3^1 = op_1$ and picks the first credentials for generating the ballot b_3 for op_3. Since $\mathcal{A}$ picked the fake credentials, the ballot b_3 in the **Tally** 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 $\mathcal{A}$'s intention was to vote for options op_3, op_1, op_3, his real choices are op_3, op_2, op_1. If privacy holds, $\mathcal{A}$ is unable to understand whether he voted for U_i^0 and the challenger for an option U_i^1 or vice versa.

Example 2

$\mathcal{C}$'s coin $a = 0$, $\mathcal{A}$ sent the challenger options U_i^0, U_i^1 and picked the credentials s_i^0 for voters V_0, V_3, V_5 and the credentials s_i^1 for the remaining voters V_1, V_2, V_4 to vote for an option U_i^0 for all $i \in [0, 5]$. Since the coin $a = 0$, the left credentials s_i^0 are fake. That means, adversarial ballots are actually for options $U_0^1, U_1^0, U_2^0, U_3^1, U_4^0, U_5^1$ instead of $U_0^0, U_1^0, U_2^0, U_3^0, U_4^0, U_5^0$ as it wanted. On the other hand, challenger's ballots are for options $U_0^0, U_1^1, U_2^1, U_3^0, U_4^1, U_5^0$. Suppose that the sum of all challenger's ballots is denoted as $\tau_c = f(U_0^0) + f(U_1^1) + f(U_2^1) + f(U_3^0) + f(U_4^1) + f(U_5^0)$ and adversarial ones as $\tau_a = f(U_0^1) + f(U_1^0) + f(U_2^0) + f(U_3^1) + f(U_4^0) + f(U_5^1)$. If τ_c and τ_a both give the same result, we can remove all challenger's ballots and compute the election result as an output of the **Result** protocol over τ_a. In this case $\mathcal{C}$ would not leak the coin a. Equivalence of τ_c and τ_a means that $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$.

Definition 2.3.1 (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 $\mathcal{A}$:

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

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

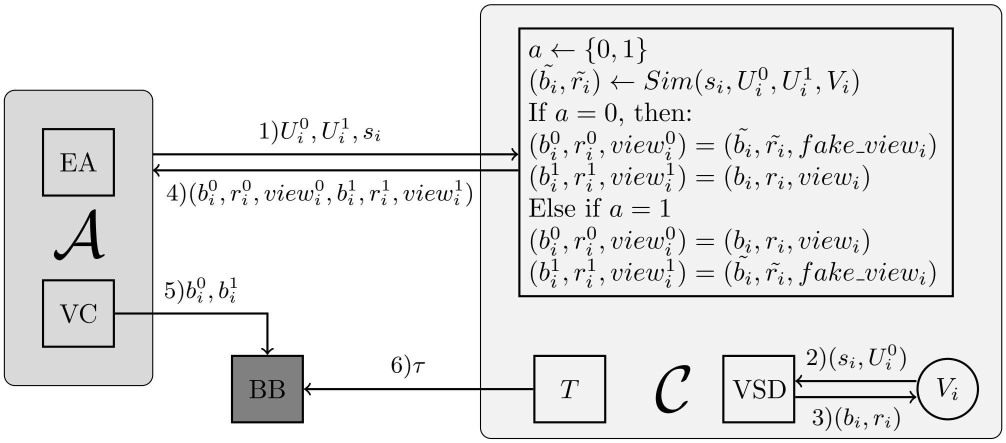


Figure 2-2: $G^{\mathcal{A},Sim}_{t-priv,\mathrm{VSD}}(1^\lambda, n, m)$

Even though it is possible to achieve voter privacy for trusted VSD **only** (by making the T unable to decrypt individual votes), such case is rather extreme. The definition of voter privacy below is given for trusted VSD and T. However, we identify changes that would transform the given definition into **only** VSD is trusted case.

In the game $G^{\mathcal{A},Sim}_{t-priv,\mathrm{VSD},\mathrm{T}}(1^\lambda, n, m)$, an adversary $\mathcal{A}$ operates on behalf of the all corrupted voters and all corrupted entities, such as EA and VC. $\mathcal{C}$ plays on behalf of VSD, T and all honest voters. BB is completely passive and represents a publicly accessible database.

1) $\mathcal{A}$ picks and sends options U_i^0, U_i^1 to the challenger $\mathcal{C}$. After that $\mathcal{A}$ provides a voter V_i with credentials s_i and 2) schedules the **Cast** protocol with $\mathcal{C}$ on behalf of V_i. 3) $\mathcal{C}$ generates a real ballot, receipt and view and uses Sim to create the fake ones. 4) At the end $\mathcal{C}$ responses with two ballots, receipts and views $b_i^0, r_i^0, view_i^0, b_i^1, r_i^1, view_i^1$, where the order of real and fake output is determined according to a coin a. 5) If $\mathcal{A}$ posts a ballot on the BB, $\mathcal{C}$ posts the other ballot*. 6) When $\mathcal{A}$ stops the election, $\mathcal{C}$ posts the tally τ **

Remarks:

Denote the list of honest voters for which $\mathcal{A}$ chooses to post the left ballot b_i^0 as $\tilde{\mathcal{V}}^0$ and the similar list but for the right one b_i^1 as $\tilde{\mathcal{V}}^1$.

*$\mathcal{C}$ should post to the BB the remaining ballot, otherwise $\mathcal{A}$ would guess the coin a by simply checking the result of an election. $\mathcal{A}$ is allowed to pick any ballot form the pair b_i^0, b_i^1 and play as many rounds as it likes. Since lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ not necessarily sums to the same result as lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$, the adversary would trivially break privacy. To prevent it, we add the challenger's ballots to the BB and compute the combined tally, namely $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1})$. However, if lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed up to the same result i.e. $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, we remove the challenger's ballots and compute the actual tally.

** The tally τ is posted only if, for every correctly formed adversarial ballot, there is a corresponding challenger's ballot posted on the BB. $\mathcal{A}$ may not post some ballots, however, if it does, then the corresponding challenger's ballot must be posted as well. Otherwise, $\mathcal{A}$ wins the *Voter Privacy* game by simply checking the announced result. $\mathcal{C}$ removes all its ballots and computes tally for the adversarial ballots if and only if lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ produce the same result $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$. In this case, the announced result would be $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1})$ if challenger's coin $a = 0$ or $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$ otherwise.

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

1. $\mathcal{A}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, chooses a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$ and the set of allowed candidates' selections $\mathcal{U}$. $\mathcal{A}$ starts an election using $\mathcal{V}, \mathcal{P}, \mathcal{U}$ as input parameters.

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

3. $\mathcal{A}$ sends to $\mathcal{C}$ options $U_i^0, U_i^1 \in \mathcal{U}$, where U_i^0 is an option for the real pair ballot and receipt and U_i^1 is an option for the fake one. 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 both simulated and real ballots, receipts and views $(b_i^0, r_i^0, view_i^0)(b_i^1, r_i^1, view_i^1)$ s.t.:
   $$\begin{cases} \text{if } a = 0, \ (b_i^0, r_i^0, view_i^0) = (\tilde{b_i}, \tilde{r}_i, fake\_view_i) \text{ and } (b_i^1, r_i^1, view_i^1) = (b_i, r_i, view_i) \\ \text{else } (b_i^0, r_i^0, view_i) = (b_i, r_i, view_i) \text{ and } (b_i^1, r_i^1, view_i^1) = (\tilde{b_i}, \tilde{r}_i, fake\_view_i) \end{cases}$$
   where the pair (b_i, r_i) is the ballot and receipt for an adversarial option U_i^0 and $(\tilde{b_i}, \tilde{r}_i)$ is the ballot and receipt for U_i^1 option generated via the simulator Sim.

   – If $\mathcal{A}$ posts a ballot on BB, $\mathcal{C}$ posts the remaining ballot.

4. Denote the list of honest voters for which $\mathcal{A}$ chooses to post the ballot b_i^0 as $\tilde{\mathcal{V}}^0$ and the similar list but for ballots b_i^1 as $\tilde{\mathcal{V}}^0$. If lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed up to the same result i.e. $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, $\mathcal{C}$ removes all its ballots and executes the **Tally** protocol on the cleared BB. Otherwise, $\mathcal{C}$ executes the **Tally** protocol on the BB that contains ballots for options U_i^0 and U_i^1.

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 $\tilde{\mathcal{V}} = \mathcal{V} \setminus \mathcal{V}_{corr}$. 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. $\mathcal{A}$ can control T and perform the **Tally** procedure. However, in such case, it should be impossible for T to learn the underline vote for any individual ballot or compute the result for BB that does not contain all ballots b_i^0, b_i^1. The only exception is the case when lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed up to the same tally as lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$, for this situation $\mathcal{A}$ may perform **Tally** operation without challenger's ballots.

Definition 2.4.1 (Privacy: VSD and T are honest)**.** The e-voting system Π achieves voter privacy in case of honest VSD and T, for at most t corrupted voters , if there is a PPT simulator Sim such that for any PPT adversary $\mathcal{A}$:

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

2.5 Comparison with existing definitions

The right to an anonymous vote is one of the main requirement for any e-voting system. However, not all information related to an election is considered to be private. For example, some data is collected for public purposes: voter's name and address, whether he participated in elections for the last decade or not etc. One thing that is universally private everywhere is the content of the ballot, namely how you voted.

Privacy of votes was the subject of great attention in last decades. There are many different approaches that try to capture privacy of voters. A strong security definition can be given as 1) an ideal functionality, 2) based on the entropy or 3) in the game-based form. The first approach, while being very powerful, results in quite difficult security proofs for real protocols. The second approach implies a weaker security since for it to be secure, the secret's distribution should possess high entropy at least from an adversary's view. The third approach provides the good security guarantees and allows to construct relatively simple security proofs compared to the first simulation-based approach.

Typically the game-based security definitions are specified as an interaction between an "adversary" ($\mathcal{A}$) and a "challenger" ($\mathcal{C}$). All honest entities are controlled by $\mathcal{C}$, the rest is played by $\mathcal{A}$. The adversarial goal is to correctly guess a particular piece of hidden information, usually a bit. The game normally specifies what $\mathcal{A}$ can do and what $\mathcal{C}$'s response would be. The security is defined in the form of the following statement: for all adversaries, the probability of winning the game does not exceed some fixed threshold.

A highly helpful survey on game-based privacy definitions has been done by Bernhard et al. in [8], where they analyse weaknesses of all existing definitions and propose their notion for privacy BPRIV. Based on the results of this survey we compare our privacy definition only with Helios ballot privacy, BPRIV and Demos Privacy definitions.

2.5.1 Helios. Ballot privacy.

Ballot privacy ensures that during protocol execution, no information about the cast votes, beyond what the result of the election leaks, is revealed. In some works, ballot privacy defined even stronger: "a voter's vote is not revealed to anyone" [9]. However, in most cases ballot privacy targets specifically vote-casting procedure and nothing else.

Informally, ballot privacy implies that an adversary in control of arbitrarily many voters cannot distinguish between real ballots and fake ballots that are ballots for some fixed vote ϵ chosen by the 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 2.5.1 (Ballot privacy for Helios [9])**.** 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 Helios 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, the final result was decrypted correctly. 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' contain 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 against 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.

A slightly modified privacy definition was given by Bernhard et al in their later work [8]. The extended to global setups version of BPRIV gives a simulator control over the fake global setup, so when an adversary calls to the global setup the simulator is in charge of answering them. The ballot privacy definition BPRIV implies that election results should not contain hidden auxiliary data. The simulated setup grant to a simulator additional powers that are useful in producing valid looking proofs for false statements. However, this approach contradicts the notion of end-to-end verifiability in the standard model [31], where all entities can be malicious but still the final result can not be manipulated without a high probability of been caught. Suppose is BPRIV private, then no one can distinguish whether a global setup is fake or real, which means the malicious EA can generate global setup with a trapdoor and cook up any proofs it likes.

2.5.2 Demos privacy

The only game-based definition we are aware of that compatible with E2E Verifiability in the standard model is *Demos Voter Privacy* defined by Kiayias et al. in [31].

Demos Voter Privacy definition resembles witness indistinguishability of interactive proof system. An adversary's challenge is to distinguish between two pre-defined by the adversary $\mathcal{A}$ lists of candidate selection that are summed 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 the 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.

Definition 2.5.2 (Demos privacy definition.)**.** The game $G^{*\mathcal{A},\mathcal{S}}_{DEMOS,t-priv}(1^\lambda, n, m)$ defined as follows [31]:

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 $\mathcal{V} = \{V_1, ..., V_n\}$, chooses 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 in the modified one it is mandatory for $\mathcal{A}$ to fulfil the following requirement: lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$, $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$, $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$ that are summed up to the same tally.

Definition 2.5.3 (Modified original privacy with res. to EA and T 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 and credentials in a such way that $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, where $\tilde{\mathcal{V}}^0$ is a list of honest voters for which $\mathcal{A}$ chooses to post produced ballot and uses credentials s_i^0 and $\tilde{\mathcal{V}}^1$ is a similar list but for credentials s_i^1.

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

1. $\mathcal{A}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, chooses 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 $a \leftarrow \{0, 1\}$ to define an order according to which real and simulated credentials would be returned to $\mathcal{A}$, and starts the election on behalf of EA.

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 and 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 forwards 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.

   – $\mathcal{A}$ chooses whether to post the produced ballot b_i to the BB or not.

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) $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$ (i.e., the election result does not leak a), where $\tilde{\mathcal{V}}^0$ is the list of honest voters for which $\mathcal{A}$ chooses to post produced ballot and uses credentials s_i^0 and $\tilde{\mathcal{V}}^1$ is the similar list but for credentials s_i^1.

Definition 2.5.4 (Modified original privacy with res. to VSD and T definition.)**.** The game $G^{\mathcal{A},Sim}_{mod\_ORIG,t-priv,\mathrm{VSD},\mathrm{T}}(1^\lambda, n, m)$ is defined as follows:

1. $\mathcal{A}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, chooses a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$ and the set of allowed candidates' selections $\mathcal{U}$. $\mathcal{A}$ starts an election using $\mathcal{V}, \mathcal{P}, \mathcal{U}$ as input parameters.

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

3. $\mathcal{A}$ sends to $\mathcal{C}$ options $U_i^0, U_i^1 \in \mathcal{U}$, where U_i^0 is an option for the real ballot and receipt and U_i^1 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 a pair of simulated and real ballot,receipt and view $(b_i^0, r_i^0, view_i^0)(b_i^1, r_i^1, view_i^1)$ s.t.:
   $$\begin{cases} \text{if } \ a = 0, \ \ (b_i^0, r_i^0, view_i^0) = (\tilde{b_i}, \tilde{r_i}, fake\_view_i) \ \text{ and } \ (b_i^1, r_i^1, view_i^1) = (b_i, r_i, view_i) \\ \text{else } \ (b_i^0, r_i^0, view_i) = (b_i, r_i, view_i) \ \text{ and } \ (b_i^1, r_i^1, view_i^1) = (\tilde{b_i}, \tilde{r_i}, fake\_view_i) \end{cases}$$
   where the pair (b_i, r_i) is the ballot and receipt for an adversarial option U_i^0 and $(\tilde{b_i}, \tilde{r_i})$ is the ballot and receipt for U_i^1 option generated via the simulator Sim.

   – If $\mathcal{A}$ posts a ballot on BB, $\mathcal{C}$ posts the remaining ballot.

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 $\tilde{\mathcal{V}} = \mathcal{V} \setminus \mathcal{V}_{corr}$. 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) $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, where $\tilde{\mathcal{V}}^0$ is the list of honest voters for which $\mathcal{A}$ chooses to post the left ballots b_i^0 and $\tilde{\mathcal{V}}^1$ is the similar list but for right ballots b_i^1.

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.

Consider the *Modified Voter Privacy* game in case when EA and T are trusted. We can construct an adversary $\mathcal{A}$, that exploits $\mathcal{B}$, and wins the game against *Modified Voter Privacy*.

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

1. $\mathcal{B}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, chooses 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 i.e. $view_i$. 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 $\mathcal{C}$ plays on behalf of EA. 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. Since *Demos Privacy* implies that $f(\langle \mathcal{U}_l^0 \rangle_{V_l \in \tilde{\mathcal{V}}}) = f(\langle \mathcal{U}_l^0 \rangle_{V_l \in \tilde{\mathcal{V}}})$ and $\mathcal{A}$ always posts all ballots and always uses the first credentials s_i^0, then $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, where $\tilde{\mathcal{V}}^1$ is empty. $\mathcal{A}$ 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. 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. 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 winning the modified privacy game against the challenger $\mathcal{C}$ 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 Voter Privacy* for the case of trusted EA and T.

We will show, that *Demos Privacy* is also weaker than *Modified Voter Privacy* for the case of trusted VSD and T. 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 Voter Privacy* in case of trusted VSD and T.

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

1. $\mathcal{B}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, chooses a list of candi-

dates $\mathcal{P} = \{P_1, ..., P_m\}$ and the set of allowed candidates' selections $\mathcal{U}$. It provides $\mathcal{A}$ with $\mathcal{V}, \mathcal{P}, \mathcal{U}$.

2. $\mathcal{A}$ starts the election on behalf of EA.

3. $\mathcal{C}$ 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 **Cast** protocols. 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 $\mathcal{A}$ provides 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 VSD, 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 $\mathcal{C}$. $\mathcal{C}$ returns $(r_i^0, b_i^0, view_i^0, r_i^1, b_i^1, view_i^1$ in order defined by the coin a:
   $$\begin{cases} \text{if } \ a = 0, \ \ (r_i^0, b_i^0, view_i^0, r_i^1, b_i^1, view_i^1) = (\tilde{r}_i, \tilde{b}_i, fake\_view_i, r_i, b_i, view_i) \\ \text{else } \ (r_i^0, b_i^0, view_i^0, r_i^1, b_i^1, view_i^1) = (r_i, b_i, view_i, (\tilde{r}_i, \tilde{b}_i, fake\_view_i) \end{cases}$$
   $\mathcal{A}$ always sends the first receipt and view $r_i^0, view_i^0$ to $\mathcal{B}$. If $\mathcal{A}$ indeed picked the real pair, r_i^0, b_i^0 would correspond to the option U_i^0 and $view_i^0$ would be real. Otherwise, r_i^0, b_i^0 would be the ballot and receipt for the option U_i^1 and $view_i^0$ would be fake. The adversary $\mathcal{B}$ is allowed to observe the network trace of the **Cast** protocol where $\mathcal{C}$ plays the role of V_l and VSD and $\mathcal{A}$ plays on behalf of EA and BB.

   (a) $\mathcal{C}$ performs the **Tally** protocol playing the role of T. Since *Demos Privacy* implies that $f(\langle \mathcal{U}_l^0 \rangle_{V_l \in \tilde{\mathcal{V}}}) = f(\langle \mathcal{U}_l^0 \rangle_{V_l \in \tilde{\mathcal{V}}})$ and $\mathcal{A}$ always posts the first ballot, then $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, where $\tilde{\mathcal{V}}^1$ is empty. $\mathcal{A}$ and $\mathcal{B}$ are allowed to observe the network trace of that protocol.

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

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

If the challenger's $\mathcal{C}$ coin $a = 0$, then the first values $r_i^0, b_i^0, view_i^0$ are always fake. Since $\mathcal{A}$ always uses $r_i^0, b_i^0, view_i^0$, $\mathcal{B}$ 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. This case corresponds to the coin $b = 1$ in the *Demos Privacy* game.

Else if $\mathcal{C}$'s coin $a = 1$, values $r_i^0, b_i^0, view_i^0$ are always real. As a result of the **Cast** protocol execution $\mathcal{B}$ 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. 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 challenger $\mathcal{C}$ with probability more than random coin flipping, which means $\mathcal{A}$ breaks *Modified Voter Privacy* definition.

We prove that if there exist a successful adversary agains *Demos Privacy*, then there also exist an adversary who breaks the *Modified Voter Privacy*. Therefore, *Demos Privacy* implies weaker level of privacy than *Modified Voter Privacy* for both cases of collision. On the other hand, *Modified Voter Privacy* is the particular case of *Voter Privacy*. This means that if *Demos Privacy* is broken than *Voter Privacy* is also would be broken.The *Voter Privacy* game implies at least the *Demos Privacy*.

Chapter 3

Strict privacy notion

In this section, we introduce a strong privacy definition where **all** voters are corrupted but an adversary is still unable to break privacy, denoted as strict privacy. We formally define strict voter privacy via a *Strict Voter Privacy* game , denoted as $G^{\mathcal{A},Sim}_{strict,<honest\ entities>}(1^{\lambda})$, that is played between an adversary A and a challenger C.

According to the game rules, an adversary is allowed to define the election parameters, corrupt some entities, and act on behalf of all voters. $\mathcal{C}$ plays role of honest parties and returns to the adversary simulated and real view of a voter in order defined by a pre-flipped coin a (If $a = 0, \mathcal{C}$ returns $(simulated\_view, real\_view)$ and $(real\_view, simulated\_view)$ otherwise). During the interaction with $\mathcal{C}$, $\mathcal{A}$ would try to guess the coin a. The *Strict Voter Privacy* game takes as input the security parameter λ, the number of candidates m and the number of voters n, where n, m are polynomial in λ, and returns 1 or 0 depending on whether the adversary wins or not.

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 other 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 $G^{\mathcal{A},Sim}_{t-priv,<honest\ entities>}(1^\lambda)$.

The specification of trusted entities splits the Voter Privacy into different scenarios. All meaningful cases of collusion fall into two scenarios: (1) EA and T are honest and (2) VSD and T are honest. It is possible to construct e-voting scheme, that is private with respect to only trusted VSD, however, most known schemes require trusted T as well.

We will show that the notion of strict privacy is the weakest level of privacy that contradicts end-to-end verifiability. To prove the contradiction we used the E2E verifiability definition given by Kiayias et al [1] in form of the ideal functionality [3] and considered a strict voter privacy case where all voters are corrupted but an adversary is still unable to break privacy. Under this framework, we prove that it is not possible to achieve E2E verifiability in any strictly private system. Moreover, any meaningful relaxation of the strict privacy definition, leads to a notion if privacy that is feasible by some E2E verifiable e-voting system.

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

Strict Voter Privacy implies that even if all voters are corrupted, an adversary $\mathcal{A}$ is still unable to break the individual privacy and learn the actual votes. The *Strict Voter Privacy* game is similar to the *Voter Privacy* game, but instead of scheduling the **Registration** protocol for a voter V, $\mathcal{A}$ starts the protocol itself.

In the game $G^{\mathcal{A},Sim}_{strict,\mathrm{EA},\mathrm{T}}(1^\lambda, n, m)$, 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.

1) $\mathcal{A}$ picks and sends two options U_i^0, U_i^1 to the challenger $\mathcal{C}$. The first option U_i^0 is its intent, the other – the option that the challenger would use in order to fool $\mathcal{A}$*. 2) After sending options, $\mathcal{A}$ runs the **Registration** protocol with $\mathcal{C}$ on behalf of some voter V_i. 3) $\mathcal{C}$

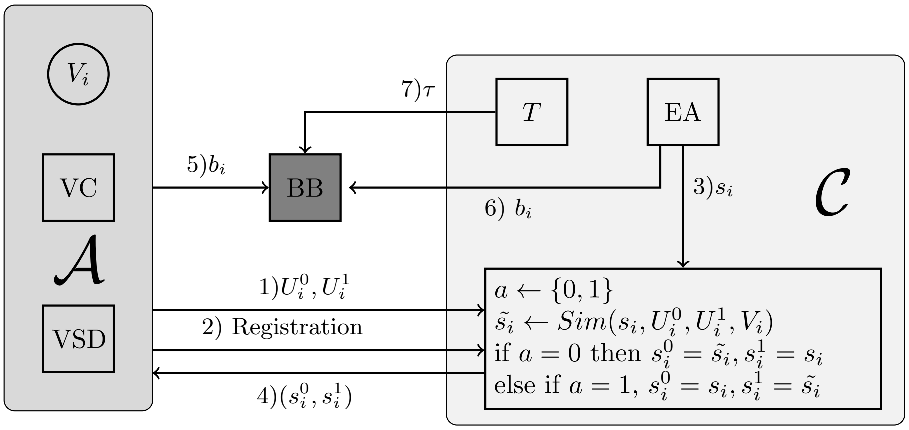


Figure 3-1: $G^{\mathcal{A},Sim}_{strict,\mathrm{EA},\mathrm{T}}(1^\lambda, n, m)$

creates fake credentials $\tilde{s}_i$ and generates real credentials s_i for the voter V_i. 4) $\mathcal{C}$ responds to the $\mathcal{A}$ with a pair of credentials s_i^0, s_i^1, where one of the credentials are real and the other were generated using the simulator Sim in a such way, that if $\mathcal{A}$ guesses right and uses the real credentials to cast a vote for U_i^0, the produced ballot and internal view of the **Cast** protocol would be real, otherwise generated ballot would correspond to the option U_i^1 and the returned view would be fake. 5) If $\mathcal{A}$ chooses to post the ballot b_i to BB, 6) $\mathcal{C}$ posts exactly the same ballot** to BB. 7) When $\mathcal{A}$ stops the election, $\mathcal{C}$ posts the tally τ ***.

Remarks:

Denote the list of voters for which $\mathcal{A}$ chooses to post produced ballot and uses credentials s_i^0 as $\tilde{\mathcal{V}}^0$ and the similar list but for credentials s_i^1 as $\tilde{\mathcal{V}}^1$.

*If $\mathcal{C}$ succeeds, $\mathcal{A}$ wouldn't be able to say whether it voted for option U_i^0 or U_i^1 and BB would contain both ballots (one for the option U_i^0 ,the other for the option U_i^1) so the tally wouldn't reveal any information. Example: if $\mathcal{A}$ picks real credential then it casts a vote for the option U_i^0 on behalf of a voter V_i, at the same time exactly the same ballot posted by $\mathcal{C}$ would correspond to the fake credentials and the option U_i^1. In this case, $\mathcal{A}$ indeed voted for U_{i_1} as it intended. Else if $\mathcal{A}$ picks the fake credentials and votes for the option U_i^0, its ballot corresponds to the real credentials and the option U_i^1. At the same time, exactly the same ballot posted by $\mathcal{C}$ corresponds to the fake credentials and the option U_i^0. So $\mathcal{A}$ voted for U_i^0. If this privacy holds, $\mathcal{A}$ has no idea what option voter V_i voted for.

** $\mathcal{C}$ should post to the BB a ballot for the remaining candidate. If $\mathcal{A}$ voted for U_i^0, $\mathcal{C}$ should vote for the U_i^1 and vice versa. Otherwise $\mathcal{A}$ would guess the coin a by simply checking the result of an election. $\mathcal{A}$ is allowed to pick any two options U_i^0, U_i^1 and play as many rounds as it likes. Since lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ not necessarily produce the same result as lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$, the adversary would trivially break privacy. To prevent it, we add the challenger's ballots to the BB and compute the combined tally, namely $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1})$. However, if lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed to the same result $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, we remove the challenger's ballots and compute the actual tally.

*** The tally τ is posted only if, for every correctly formed adversarial ballot, there is a corresponding challenger's ballot posted on the BB. $\mathcal{A}$ may not post some ballots, however, if it does, then challenger's ballot must be posted as well. Otherwise, $\mathcal{A}$ wins the *Voter Privacy* game by simply checking the announced result. All $\mathcal{A}$'s ballots are tallied based on real credentials, all $\mathcal{C}$'s ones – based on fake credentials. $\mathcal{C}$ removes all its ballots and computes tally for the adversarial ballots if and only if lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ produce the same result, i.e $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$. In this case, the result would be $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1})$ if challenger's coin $a = 0$ or $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$ otherwise.

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

1. $\mathcal{A}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, chooses 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 $a \leftarrow \{0, 1\}$ to define an order according to which real and simulated

credentials would be returned to $\mathcal{A}$, and starts the election on behalf of EA.

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 $\mathcal{A}$ runs the **Registration** protocols on behalf of all voters. For each voter $V_i \in \mathcal{V}$:

   – $\mathcal{C}$ generates real credentials s_i and 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 the order defined by the coin a: $\begin{cases} \text{if } a = 0, & (s_i^0, s_i^1) = (\tilde{s}_i, s_i) \\ \text{else} & (s_i^0, s_i^1) = (s_i, \tilde{s}_i) \end{cases}$

   – 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. 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 i.e. $view_i$. 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. 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. Denote the list of voter for which $\mathcal{A}$ chooses to post produced ballot and uses credentials s_i^0 as $\tilde{\mathcal{V}}^0$ and the similar list but for credentials s_i^1 as $\tilde{\mathcal{V}}^1$. If lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed to the same result $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, $\mathcal{C}$ removes all its ballots and executes the **Tally** protocol on the cleared BB. Otherwise, $\mathcal{C}$ executes the **Tally** protocol on the BB that contains adversarial and its own ballots.

5. Finally, $\mathcal{A}$ using all information collected above (including the contents of the BB)

outputs a bit a^*

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

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

1. The real credentials and U_i^0 option should give ballot and receipt, which are identical to the ballot and receipt produced for the fake credentials and U_i^1 option.

2. The fake credentials and U_i^0 option should give ballot and receipt, which are identical to the ballot and receipt produced for the real credentials and U_i^1 option.

Definition 3.1.1 (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 $\mathcal{A}$:

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

3.2 VSD and T are honest: $G^{\mathcal{A},Sim}_{strict,\mathrm{VSD},\mathrm{T}}(1^\lambda, n, m)$

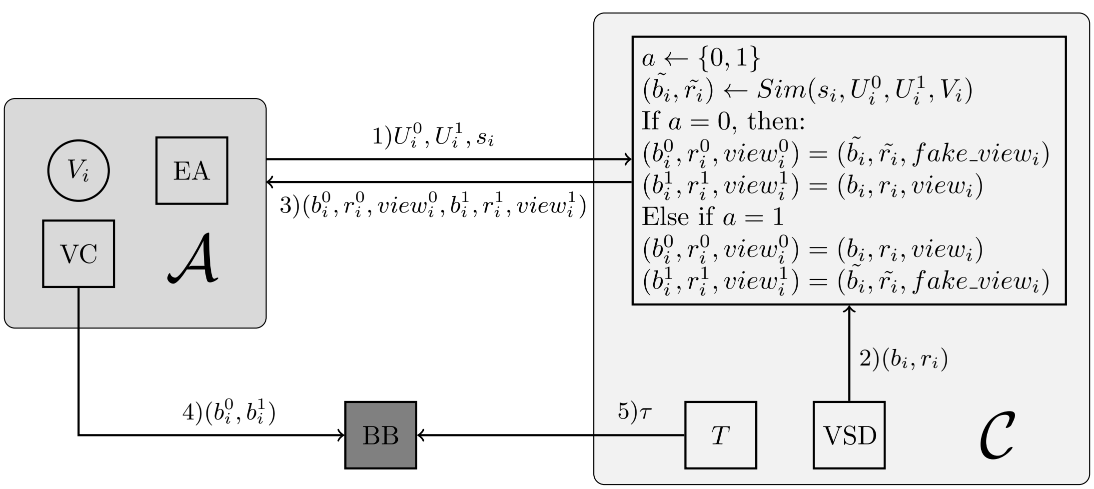


Figure 3-2: $G^{\mathcal{A},Sim}_{strict,\mathrm{VSD},\mathrm{T}}(1^\lambda)$

In *Strict Voter Privacy* game, we do not consider the case where privacy of voters relies on trusted VSD **only** since the case is rather extreme and rare. The definition of strict voter privacy below is given for trusted VSD and T. However, we identify changes that would transform the given definition into **only** VSD is trusted case.

In the game $G^{\mathcal{A},Sim}_{strict,\mathrm{VSD},\mathrm{T}}(1^\lambda, n, m)$ there is no honest voters, instead an adversary $\mathcal{A}$ operates on behalf of the all voters, EA and VC. $\mathcal{C}$ plays on behalf of VSD and T. BB is completely passive and represents a publicly accessible database.

1) $\mathcal{A}$ picks and sends options U_i^0, U_i^1 to the challenger $\mathcal{C}$. 2) After that $\mathcal{A}$ runs the **Cast** protocol with $\mathcal{C}$. 3) $\mathcal{C}$ generates a real ballot, receipt and view for an option U_i^0 and uses Sim and option U_i^1 to create the fake ones. 4) At the end $\mathcal{C}$ responses with a pair of ballots, receipts and views $b_i^0, r_i^0, view_i^0, b_i^1, r_i^1, view_i^1$, where the order of real and fake output is determined according a coin a. 5) If $\mathcal{A}$ posts a ballots on the BB, $\mathcal{C}$ posts the other ballot*. 6) When $\mathcal{A}$ stops the election, $\mathcal{C}$ posts the tally τ **

Remarks:

Denote the list of voters for which $\mathcal{A}$ chooses to post the left ballot b_i^0 as $\tilde{\mathcal{V}}^0$ and the similar list but for the right one b_i^1 as $\tilde{\mathcal{V}}^1$.

*$\mathcal{C}$ should post the remaining ballot to the BB, otherwise $\mathcal{A}$ would guess the coin a by simply checking the result of an election. $\mathcal{A}$ is allowed to pick any ballot form the pair b_i^0, b_i^1 and play as many rounds as it likes. Since lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ not necessarily sums to the same result as lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$, the adversary would trivially break privacy. To prevent it, we add the challenger's ballots to the BB and compute the combined tally, namely $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1})$. However, if lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed up to the same result i.e. $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, we remove the challenger's ballots and compute the actual tally.

** The tally τ is posted only if, for every correctly formed adversarial ballot, there is a corresponding challenger's ballot posted on the BB. $\mathcal{A}$ may not post some ballots, however, if it does, then the corresponding challenger's ballot must be posted as well. Otherwise, $\mathcal{A}$ wins the *Voter Privacy* game by simply checking the announced result. $\mathcal{C}$ removes all its ballots and computes tally for the adversarial ballots if and only if lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ produce the same result $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$. In this case, the announced result would be $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1})$ if challenger's coin $a = 0$ or $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$ otherwise.

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

1. $\mathcal{A}$ on input $1^\lambda, n, m$ defines a set of voters $\mathcal{V} = \{V_1, ..., V_n\}$, chooses a list of candidates $\mathcal{P} = \{P_1, ..., P_m\}$ and the set of allowed candidates' selections $\mathcal{U}$. $\mathcal{A}$ starts an election using $\mathcal{V}, \mathcal{P}, \mathcal{U}$ as input parameters.

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

3. $\mathcal{A}$ sends to $\mathcal{C}$ options $U_i^0, U_i^1 \in \mathcal{U}$, where U_i^0 is an option for the real ballot and receipt and U_i^1 is an option for the fake ones. After that, $\mathcal{A}$ and $\mathcal{C}$ engage in an interaction where $\mathcal{A}$ runs the **Cast** protocols of all voters which may run concurrently. For each voter $V_i \in \mathcal{V}$:

   – $\mathcal{C}$ provides $\mathcal{A}$ with a pair of simulated and real ballot,receipt and view $(b_i^0, r_i^0, view_i^0)(b_i^1, r_i^1, view_i^1)$ s.t.:
   $$\begin{cases} \text{if } \ a = 0, \ \ (b_i^0, r_i^0, view_i^0) = (\tilde{b_i}, \tilde{r_i}, fake\_view_i) \ \text{ and } \ (b_i^1, r_i^1, view_i^1) = (b_i, r_i, view_i) \\ \text{else } \ (b_i^0, r_i^0, view_i) = (b_i, r_i, view_i) \ \text{ and } \ (b_i^1, r_i^1, view_i^1) = (\tilde{b_i}, \tilde{r_i}, fake\_view_i) \end{cases}$$
   where the pair (b_i, r_i) is the ballot and receipt for an adversarial option U_i^0 and $(\tilde{b_i}, \tilde{r_i})$ is the ballot and receipt for U_i^1 option generated via the simulator Sim.

   – If $\mathcal{A}$ posts a ballot on BB, $\mathcal{C}$ posts the remaining ballot.

4. Denote the list of honest voters for which $\mathcal{A}$ chooses to post the ballot b_i^0 as $\tilde{\mathcal{V}}^0$ and the similar list but for ballots b_i^1 as $\tilde{\mathcal{V}}^0$. If lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}, \langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed up to the same result i.e. $f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}) + f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) = f(\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}) +$

$f(\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0})$, $\mathcal{C}$ removes all its ballots and executes the **Tally** protocol on the cleared BB. Otherwise, $\mathcal{C}$ executes the **Tally** protocol on the BB that contains ballots for options U_i^0 and U_i^1.

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

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

***Remark:**

$\mathcal{A}$ can control T and perform the **Tally** procedure. However, in such a case, it should be impossible for T to learn the underline vote for any individual ballot or compute the result for BB that does not contain all ballots b_i^0, b_i^1. The only exception is the case when lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ are summed up to the same tally as lists $\langle \mathcal{U}_i^0 \rangle_{V_i \in \tilde{\mathcal{V}}^1}$ and $\langle \mathcal{U}_i^1 \rangle_{V_i \in \tilde{\mathcal{V}}^0}$, for this situation $\mathcal{A}$ may perform **Tally** operation without challenger's ballots.

Definition 3.2.1 (Strict privacy: VSD and T are honest)**.** The e-voting system Π achieves strict voter privacy in case when VSD and T are trusted, if there is a PPT simulator Sim such that for any PPT adversary $\mathcal{A}$:

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

3.3 Strict Privacy contradicts E2E Verifiability.

As we discuss in section 2.5.2, *Demos Privacy* implies the weaker security than *Voter Privacy*, therefore the same holds for strict privacy notion. However, we use our Strict Privacy definition instead of strict version of Demos Privacy to prove that "Any system Π that satisfies the definition of strict privacy, is "receipt free" but not E2E Verifiable", because we need privacy guarantees for the different tallies case. As for E2E Verifiability, we use the ideal functionality designed by V. Teague, A. Kiayias, T. Zacharias et al. [3] that captures the E2E Verifiability definition by Kiayias et. al. The paper, where the ideal functionality for end-to-end verifiable elections is defined, currently is under submission, therefore we do not list the functionality here.

The Real and Ideal executions, in this case, are shown in figure 6.
Security requires the 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.

Theorem 1 (Strict Privacy contradicts E2E Verifiability)**.** 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 $\mathrm{EXEC}_{\mathcal{Z},\mathcal{S}}^{\mathcal{F},\mathcal{G}_{BB}} \not\approx \mathrm{EXEC}_{\mathcal{Z},\mathcal{A}}^{\Pi,\mathcal{G}_{BB}}$

3.3.1 Part 1: E2E Verifiability vs Strict privacy with respect to EA and T.

Suppose there is a system Π, which is strictly private in case "EA is corrupted, but VSD and T are 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:

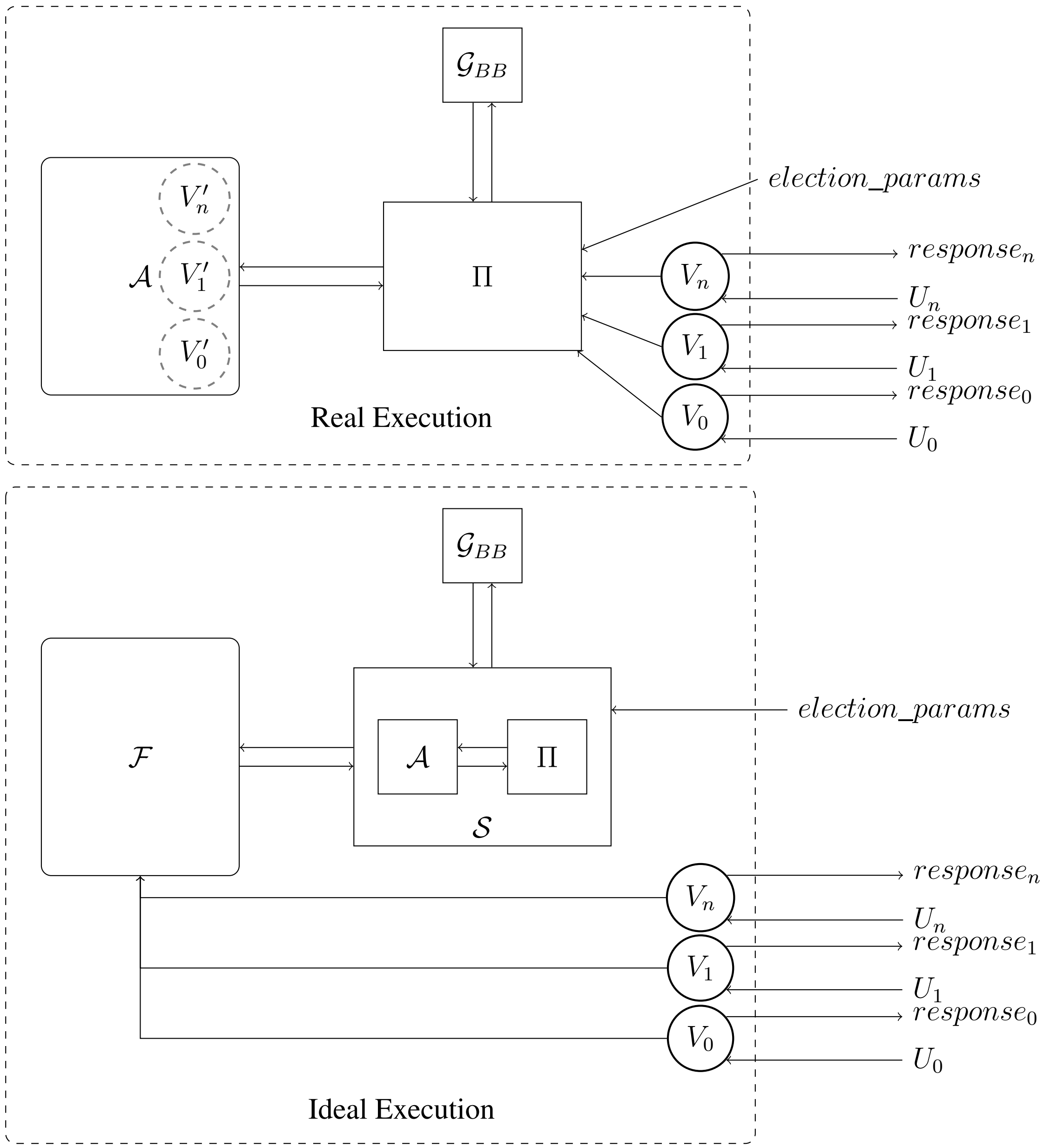


Figure 3-3: Real and Ideal Execution

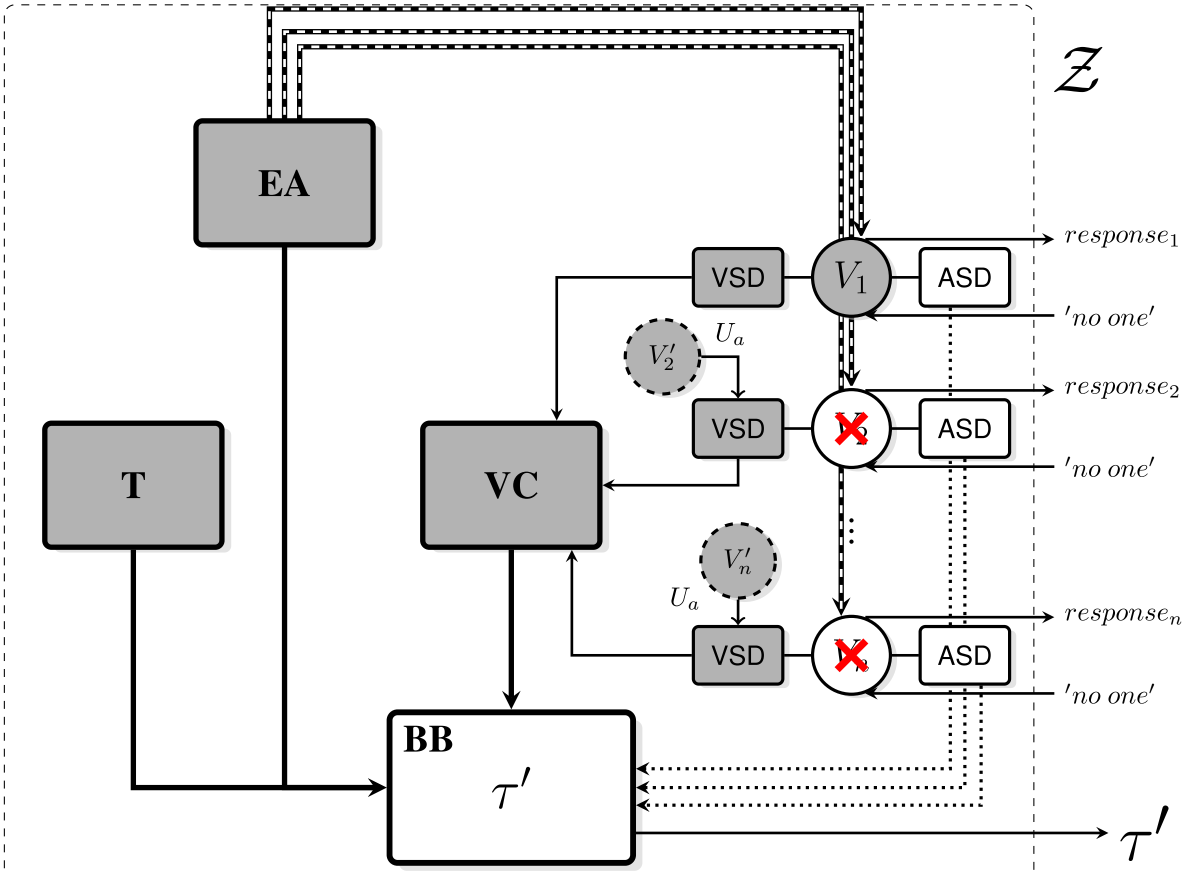


Figure 3-4: $\mathcal{A}$

$\mathcal{Z}$:

– 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.

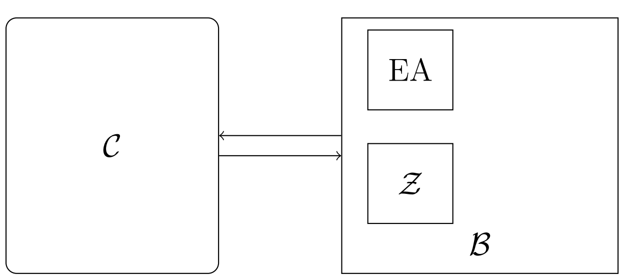
– If the number of successful verification responses equal to the number of voters and in all responses provided by a voter tally τ' is equal to the $\mathcal{G}_{BB}$'s tally τ return 1. Otherwise return 0.

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 $Pr[G^{\mathcal{D},Sim}_{strict,\mathrm{VSD,T}}(1^{\lambda,n,m}) = 1] = \frac{1}{2} + \alpha$ where α is negligible. In case of corrupted EA, $\mathcal{C}$ returns a real pair of ballot and receipt and a real view and also a simulated via Sim pair and fake view. 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_a option.

During the Real Execution a voter can either accept or reject the cast ballot. Rejection is possible only if he detects that the returned receipt, ballot and view 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, 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 $\Pr[\mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 1] = \Pr[all\ V_i\ accept] = \prod_{i=0}^{n}(1 - \Pr[V_i\ rejects]) \geq (1-\alpha)^n = 1 - n\alpha = 1 - \beta$, where $\beta = n\alpha$ is negligible. Thus, $\Pr[\mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] = 1 - \Pr[\mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 1] \leq \beta$, where β is negligible.

Suppose for the sake of contradiction that $\Pr[\mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] \geq \beta$, where β is non-negligible. This means that at least one voter rejects the receipt. 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 every voter to vote for the blank option $\{V_i, \text{`}no\ one'\}$ 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 $(V_i, \text{`}no\ one', U_{\mathcal{Z}_i})$ 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 $\Pr[\mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] \geq \beta$, 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 $Pr[\mathcal{B} \rightarrow 0 | a = 0] = \Pr[\mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0] \geq \beta$, 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 $\Pr[G^{\mathcal{B}}_{strict}(1^\lambda) = 1] = \Pr[\mathcal{B} \to 0|a = 0]\Pr[a = 0] + \Pr[\mathcal{B} \to 1|a = 1]\Pr[a = 1] = \Pr[\mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},\mathcal{A}} = 0]\Pr[a = 0] + \Pr[\mathrm{EXEC}^{\Pi,\mathcal{G}_{BB}}_{\mathcal{Z},honest} = 1]\Pr[a = 1] \geq \frac{1}{2}\beta + \frac{1}{2} - negl(\lambda)$, 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.

3.3.2 Part 2: E2E Verifiability vs Strict privacy with respect to VSD and T.

Suppose there is 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 $(s'_i, \text{'}no\ one')$ are identical to the ballot and receipt produced using real credentials for an option U_a: $b, r \leftarrow (s'_i, \text{'}no\ one')$ 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.

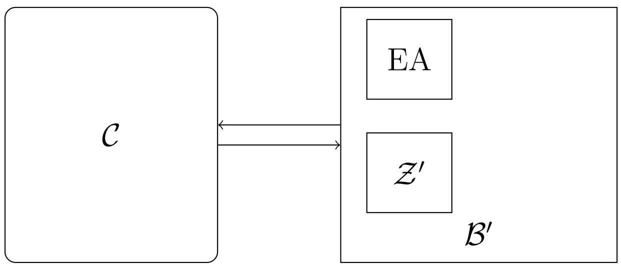
– If the number of successful verification responses equal to the number of voters and in all responses provided by a voter tally τ' is equal to the $\mathcal{G}_{BB}$'s tally τ return 1. Otherwise return 0.

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 $|\Pr[G^{\mathcal{A},Sim}_{strict,\mathrm{EA}}(1^{\lambda}, n, m) = 1] - \frac{1}{2}| = \alpha$ 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 option. 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 $\Pr[\mathrm{EXEC}^{\Pi',\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{A}'} = 1] = \Pr[all\ V_i\ accept] = \prod_{i=0}^{n}(1 - \Pr[V_i\ rejects])$. Suppose, the probability of rejection by V_i is $\Pr[V_i\ rejects] = \zeta$. Thus, $\Pr[\mathrm{EXEC}^{\Pi',\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{A}'} = 1] = (1-\zeta)^n = 1 - n\zeta = 1 - \zeta'$.

Suppose that ζ' is not negligible. This means that $\Pr[\mathrm{EXEC}^{\Pi',\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{A}'} = 0] = 1 - \Pr[\mathrm{EXEC}^{\Pi',\mathcal{G}_{BB}}_{\mathcal{Z}',\mathcal{A}'} =$

$1] = \zeta'$, 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 $\{V_i, \text{'}no\ one'\}$ 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 $\mathcal{C}$ $(V_i, \text{'}no\ one', U_a)$.

– $\mathcal{C}$ sends back s_0, s_1.

– $\mathcal{B}'$ provides voters with credentials s_0 and uses $(s_0, \text{'}no\ one')$ 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.

Chapter 4

Blind signature e-voting scheme

The idea of obtaining a proof of membership and voting anonymously is not new and had been considered the best possible solution for providing privacy to voters. However most approaches that based on group signatures [43], [4] are traceable because it requires an additional tag that revivals the user's identity in case of exceeding the allowed number of cast votes.

A blind signature, on the other hand, makes it absolutely impossible to link a particular voter and his ballot based on published after or during an election information or leakage of authentication data. It is on of the most popular techniques in e-voting schemes for ensuring confidentiality of the voter's ballot. Blind signatures have been used in many e-voting schemes designed last decades [29], [28], [23]. The main feature of the blind signature concept is that it allows to combine two contradictory ideas 1) the authorization of voters and 2)the anonymity of them. It allows to achieve such level of privacy that no one including EA and T can link the voter and his ballot.

We present an e-voting scheme that is based on the blind signature scheme and the ElGamal encryption, that is Voter Private with respect to trusted EA and T but not E2E Verifiable in the standard model. We argue that any e-voting scheme that keeps anonymous ballots is not E2E verifiable in the standard model.

4.1 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. During the election process, the *Signing server* plays the roles of the EA. Trustee T is a separate entity that has a secure channel to communicate with EA, security of communication is supported by HTTPS. VSDs are realised by Javascript 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.

4.2 Setup and parameters

Throughout the paper, it is assumed that a group of n voters $\mathcal{V} = \{V_1, \ldots, V_n\}$ 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.

It is assumed 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 the cryptographic hash-function that outputs an l_e-bit number e is used. 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 [26] suggests to use $l_s = 80$, since this value is large enough to ignore the off chance that $|a + r| > |a| + l_s$.

4.3 Syntax

An e-voting system Π is a tuple 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 $s_1, \dots, s_n$, 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. The public data is defined as $Pub = \langle PubEA, PubT \rangle$.

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.

4.4 Building blocks

4.4.1 Blind signature

In cryptography digital signature allows to 'sign' massage so everyone can verify the validity of the signature, but no one can produce a *new* signature for some new message. One variation of digital signatures is the blind signature scheme, that is the basic signature with the additional requirement that a signer should 'sign' messages without knowing its content. The concept of blind signing was introduced by David Chaum in [12].

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 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 even an adversary $\mathcal{A}$ with unlimited computing power can not break the blind signature scheme because $\mathcal{A}$ simply does not have enough information to break the encryption.

Also, Chaum's blind signature satisfies the blindness property [12] and the non-forgeability [39], [1] 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.

One can implement the blind signature scheme with almost any common public key encryption scheme. For our scheme we use one of the simplest blind signature schemes, that is based on RSA encryption.

4.4.2 Homomorphic Integer Commitment and Homomorphic Cryptosystem

There are only a few homomorphic integer commitment schemes [20], [18], [25] and they are quite similar in structure. For the blind signature scheme we use the commitment scheme described in [27]: "A modulus n as a product of two safe primes and random generators $g_1, \ldots, g_k, h$ of QR_n are chosen. In order to commit to integers $m_1, m_2, \ldots, m_k$ using randomness r, it is necessary to 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 $r \leftarrow \{0,1\}^{l_r}$.

To prove soundness and knowledge in protocols, we need a root extraction property. This property basically says that "if a ciphertext raised to a nontrivial exponent encrypts 1, then the ciphertext itself encrypts 1" [27]. For the homomorphic encryption generalisation of this property can be formalized as follows [26]:

Definition 4.4.1 (Root extracting property [26])**.** 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 [21]. 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 [27].

The use of homomorphic tallying scheme for e-voting is not new, it was originally introduced by Cramer et al. [17] in 1997. From the E2E Verifiability point of view, the major benefit of using homomorphic tallying is that the proof of correct tallying and decryption is simplified and can be publicly verified if all encrypted votes are public [38]. This means that tallied as recorder verification can be simplified and re-checked by anyone. Moreover, the aggregated ciphertext and the corresponding plaintext cannot be linked back to individual votes. Publishing encrypted ballots can leas to long-term privacy risks, for example, the encryption may be broken after a few decades. One may mitigate the risks by limiting the access to BB, where all encrypted ballots are stored. However, for the *Blind signature scheme* we prefer to publish all encrypted ballots to achieve public verifiability.

If tallied as recorder can be checked simply by anyone, then the only verifiability feature still missing (except for cast as intended check) is to verify that all ballots are correctly formed and contains no more than one vote per candidate. This means that ballots should be submitted along with the zero-knowledge cryptographic proof of correctness.

4.4.3 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 there is a signature σ for a hash value over message x $HASH(x)$. Now a prover want to convince a verifier that he knows the signature without revealing value σ. The prover chooses a random variable A and encrypt it using the public key $pkey = (e, N)$. Then he computes a product S of the encrypted value and the 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)

Proof of the signature knowledge:

$NIZK[(x, \sigma) : \sigma = x^d \bmod N]$

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.

4.4.4 Proving that a ciphertext encrypts 0 or 1

The following NIZK proof was designed by Jens Groth [26].

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 $j \in [0, m-1]$.

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

The variables a_j and Δ are hidden using the standard techniques $\boxed{a_j} = ea_j + r_{a_j}$ and $\boxed{\Delta} = e\Delta + r_\Delta$. In the verification, the verifier 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 the prover committed to $\Delta = 0$, which is exactly what needs to be proven.

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 the prover committed to.

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

The proof of correctness:

$NIZK[(v, \rho, a_0, \ldots, a_{m-1}) : C = Encr(v; \rho)$ 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 $c^e c_r = com(\boxed{a_0}, \ldots, \boxed{a_{m-1}}, \boxed{\Delta}; \boxed{r})$

The complete prove of completeness, zero-knowledge as well as soundness and knowledge of the described NIZK argument can be found here [26].

4.5 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 $e_l = (e_{1_l}, e_{2_l}, \dots, e_{m_l})$ 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

4.6 E2E Verifiability

In E2E verifiability proofs, we assume that all entities are malicious and controlled by the adversary $\mathcal{A}$ and only BB is honest, but $\mathcal{A}$ can arbitrarily change the content of the BB before executing the **Result** protocol. However, we 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 $\langle \mathcal{U}_l^{\mathcal{E}} \rangle_{V_l \in \mathcal{V} \setminus \tilde{\mathcal{V}}}$ Finally $\mathcal{E}$ outputs $\langle \mathcal{U}_l^{\mathcal{E}} \rangle_{V_l \in \mathcal{V} \setminus \tilde{\mathcal{V}}}$

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. Thus, $\mathcal{A}$ can not misinterpret results or encode more than one vote for a particular candidate. The only possible way for $\mathcal{A}$ to cheat is to somehow modify honest votes' intents.

Modification attack: $\mathcal{A}$, who controls the whole system, may attempt to modify a ballot by decrypting it, learning x, creating a new signature and voting for another candidate, however since each voter has the hash of the whole his ballot as receipt, malicious EA would be caught.

Clash attacks: This attack is statistically improbable, since x is unknown for $\mathcal{A}$ large string picked at random by a voter. If $\mathcal{A}$ modifies the pseudo-random generator in a way that a number of voters gets the same not random string, it is easily detectable since this part is done on the client side and the code is available for audition.

However EA can use abstain voters, who registered but did not cast their votes, 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. This attack can be detected only if every registered voter is assigned a specific entry in BB. Unfortunately, assigning every voter ballot id makes this system linkable.

There is a 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. This attack applies to any e-voting systems that allows voters to vote anonymously.

4.7 Implementation

The blind signature scheme system is an open source web-based public auditable e-voting system. The system consists of two main servers: *Signing Server* (SS) and *Ballot Server* (BS). The server side is written on Node.js, that uses event-based server execution procedure rather than the multithreaded execution in PHP. Basically, Node.js utilise the same even-based asynchronous technology as JavaScript. Node.js has proven itself capable of handling millions of concurrent connections and it is cross-platform. The source code is available on Github [22].

For the testing purpose we used self-singed SSL certificates for BS and SS. Certificate are used to prevent man-in-the-middle attacks and ensure that the certificate holder is really who he claims to be and usually signed by a trusted certificate authority (CA). However in the BSS proof of concept instead of obtaining real certificate signed by CA we used the openssl toolkit for generated self-sign certificates. This toolkit allows to generate a 1024-bit private key using RSA encryption and issue a certificate signing request.

Both servers SS and BS are HTTPS servers with self-signed certificate. The former server lets users to sign up for an election, check list of registered voters and sign a random credentials blindly. The later server allows users to cast their votes, check any vote in database, find their own vote based on QR code, decrypt all votes and show all cast votes.

All cryptographic primitives are implemented using Forge Javascript Cryptography Library. The Forge library implements TLS protocol and the set of cryptographic algorithms: AES (CBC), RSA, MD5, SHA-1, SHA-256 message digests, HMAC support, PKCS#5 password-based key-derivation [5].

BB is implemented as an MySQL database that contains two tables: *list* and *log*. The description of tables is given below:

Table 4.1: list

Field	Type	Null	Key	Default	Extra
number	int(5)	NO	PRI	NULL	auto_increment
id	varchar(300)	NO		NULL	
x	varchar(300)	YES		NULL	
z	varchar(300)	YES		NULL	
p	varchar(300)	NO		NULL	
psign	varchar(300)	YES		NULL	
submission_time	timestamp	NO		NULL	CURRENT_TIMESTAMP

Table 4.2: log

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
x	varchar(300)	NO		NULL	
s	varchar(300)	NO		NULL	
c	varchar(300)	NO		NULL	
submission_time	timestamp	NO		NULL	CURRENT_TIMESTAMP
vote	varchar(300)	YES		NULL	
decrvote	varchar(300)	YES		NULL	
hash	varchar(300)	NO		NULL	

Chapter 5

Conclusion

We present the generalised model of e-voting system and analyse voter privacy with respect to different cases of collision among entities of this model. The specification of trusted entities splits the Voter Privacy into different scenarios. All meaningful cases of collusion fall into two scenarios: (1) EA and T are honest and (2) VSD and T are honest. It is possible to construct e-voting scheme, that is private with respect to only trusted VSD, however, most known schemes require trusted T as well.

The approach that we used for voter privacy in this work is the following: an honest voter is allowed to have only one perfectly hidden from adversarial eyes interaction and at the end he provides the adversary with (1) the real view of the result of this interaction and (2) a simulated one via an efficient algorithm Sim called the simulator. The adversary $\mathcal{A}$ is allowed to observe a network trace of all interactions and play on behalf of corrupted entities and voters. This 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.

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 [7] 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 [31]. 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.

In this work, we perform a thorough and formal study on "locating" the critical contradiction point in the voter privacy-E2E verifiability tradeoff. As part of our analysis, in chapter 3 we introduce a strong privacy definition where voters are corrupted but an adversary is still unable to break privacy, denoted as strict privacy. We formally define strict voter privacy via a Voter Privacy game that is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. According to the game rules, an adversary is allowed to define the election parameters, corrupt a number of entities, and act on behalf of all voters. As for E2E verifiability, we apply the definition given by Kiayias et al., according to which even when all election administrators are corrupted, they can not manipulate the results without a high detection probability.

Under this framework, we prove that strict privacy even in its weakest level contradicts end-to-end verifiability. However, any meaningful relaxation of the strict privacy definition leads to a notion of privacy that is feasible by some E2E verifiable e-voting system.

Also, we design a new e-voting system based on blind signature scheme, that captures the idea of anonymous voting, where everyone votes on behalf of an eligible group of voters. We argue that any system that keeps anonymous ballots is not E2E verifiable in the standard model.

Bibliography

[1] Revised Extended Abstract, Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of Blind Digital Signatures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1294(1):150–164, 1997.

[2] Ben Adida. Helios: Web-based Open-Audit Voting. *USENIX Security Symposium*, pages 335–348, 2008.

[3] Vanessa Teague Aggelos Kiayias, Thomas Zacharias. Incoercible end-to-end verifiable elections. personal communication, 2016-2017.

[4] Man Ho Au, Joseph K. Liu, Willy Susilo, and Tsz Hon Yuen. Secure ID-based linkable and revocable-iff-linked ring signature with constant-size construction. *Theoretical Computer Science*, 469:1–14, 2013.

[5] Digital Bazaar. Forge: Javascript security and cryptography. `http://digitalbazaar.com/forge/`, 2017.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[7] Josh Benaloh and Eric Lazarus. An Attack on Verifiable Voting Systems and a Simple Mitigation. Technical report, 2011.

[8] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. *Proceedings - IEEE Symposium on Security and Privacy*, 2015-July:499–516, 2015.

[9] David Bernhard, Veronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting helios for provable ballot privacy. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6879 LNCS:335–354, 2011.

[10] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. *IEEE International Conference on Cluster Computing, SFCS*, pages 136–145, 2001.

[11] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004.

[12] David Chaum. Blind Signatures for Untraceable Payments, 1982.

[13] David Chaum. Secret-Ballot Receipts: True Voter-Verifiable Elections, 2004.

[14] David Chaum, Richard T. Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y A Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity ii: End-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Transactions on Information Forensics and Security*, 4(4):611–627, 2009.

[15] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[16] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 354–368, 2008.

[17] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi- authority election scheme. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1233:103–118, 1997.

[18] Ivan Damgard, Eiichiro Fujisaki, and Ivan 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)*, 2501:125–142, 2002.

[19] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. *Proceedings of the Computer Security Foundations Workshop*, 2006:28–39, 2006.

[20] Fujisaki Eiichiro and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. *In proceedings of CRYPTO '97, LNCS series*, 1294:16–30, 1997.

[21] Taher Elgamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[22] T. Finogina. Project title. `https://github.com/TomaF/BlindSignatureScheme`, 2017.

[23] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *Advances in Cryptology—AUSCRYPT'92*, volume 718, pages 244–251, 1993.

[24] Jens Groth. Evaluating security of voting schemes in the universal composability framework. *Applied Cryptography and Network Security*, pages 1–18, 2004.

[25] Jens Groth. Cryptography in Subgroups of Z_N*. *Theory of Cryptography*, 3378:50–65, 2005.

[26] Jens Groth. Non-interactive zero-knowledge arguments for voting. *Applied Cryptography and Network Security*, pages 467–482, 2005.

[27] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, 2010.

[28] Subariah Ibrahim, Maznah Kamat, Mazleena Salleh, and Shah Rizan Abdul Aziz. Secure E-voting with blind signature. *4th National Conference on Telecommunication Technology, NCTT 2003 - Proceedings*, (November 2014):193–197, 2003.

[29] Marius Ion and Ionuţ Posea. An Electronic Voting System Based On Blind Signature Protocol. 1:67–72, 2011.

[30] Aggelos Kiayias and Thomas Zacharias. DEMOS-2 : Scalable E2E Verifiable Elections without Random Oracles. *Ccs '15*, 2015-Octob:352–363, 2015.

[31] Aggelos Kiayias, Thomas Zacharias, and Bingsheng 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)*, 9057:468–498, 2015.

[32] Steve Kremer and Mark Ryan. Receipt-freeness: formal definition and fault attacks (Extended Abstract). pages 1–4.

[33] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. *Proceedings of the 17th ACM conference on ...*, pages 526–535, 2010.

[34] Lucie Langer, Hugo Jonker, and Wolter Pieters. Anonymity and verifiability in voting: Understanding (un)linkability. In *Proceedings of the 12th International Conference on Information and Communications Security*, ICICS'10, pages 296–310, Berlin, Heidelberg, 2010. Springer-Verlag.

[35] Huijia Lin and Rafael Pass. A Unified Framework for Concurrent Security : Universal Composability from Stand-alone Categories and Subject Descriptors. *Stoc*, pages 179–188, 2009.

[36] Yehuda Lindell. How To Simulate It - A Tutorial on the Simulation Proof Technique. *IACR Cryptology ePrint Archive*, 2016:64, 2016.

[37] Ca Neff. Practical high certainty intent verification for encrypted votes. *VoteHere document*, 2004.

[38] Arnis Parsovs. Homomorphic Tallying for the Estonian Internet Voting System. pages 1–11, 2016.

[39] David Pointcheval and Jacques Stern. Provably Secure Blind Signature Schemes. *Advances in Cryptology—ASIACRYPT'96*, 96:252–265, 1996.

[40] Ronald L Rivest. The ThreeBallot Voting System. In *in WPES 2006 — ACM Workshop on Privacy in the Electronic Society*, pages 29–40. 2006.

[41] P. Y A Ryan and S. A. Schneider. Prêt à voter with re-encryption mixes. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4189 LNCS, pages 313–326, 2006.

[42] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. *Advances in Cryptology—EUROCRYPT'95*, pages 393–403, 1995.

[43] Isamu Teranishi, Jun Furukawa, and Kazue Sako. K-times anonymous authentication. In *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, volume E92-A, pages 147–165, 2009.

[44] Wikipedia. Electronic voting. `https://en.wikipedia.org/wiki/Electronic_voting`. [Online; accessed 12-April-2016].