DEMOS-2: Scalable E2E Verifiable Elections without Random Oracles

Aggelos Kiayias*†[1], Thomas Zacharias†[1], and Bingsheng Zhang‡†[2]

[1]Dept. of Informatics and Telecommunications, University of Athens, Greece
[2]Security Lancaster Research Centre, Lancaster University, UK

October 13, 2015

Abstract

Recently, Kiayias, Zacharias and Zhang proposed a new E2E verifiable e-voting system called 'DEMOS' that for the first time provides E2E verifiability without relying on external sources of randomness or the random oracle model; the main advantage of such system is in the fact that election auditors need only the election transcript and the feedback from the voters to pronounce the election process unequivocally valid. Unfortunately, DEMOS comes with a huge performance and storage penalty for the election authority (EA) compared to other e-voting systems such as Helios. The main reason is that due to the way the EA forms the proof of the tally result, it is required to *precompute* a number of ciphertexts for each voter and each possible choice of the voter. This approach clearly does not scale to elections that have a complex ballot and voters have an exponential number of ways to vote in the number of candidates. The performance penalty on the EA appears to be intrinsic to the approach: voters cannot compute an enciphered ballot themselves because there seems to be no way for them to prove that it is a valid ciphertext.

In contrast to the above, in this work, we construct a new e-voting system that retains the strong E2E characteristics of DEMOS (but against computational adversaries) while completely eliminating the performance and storage penalty of the EA. We achieve this via a new cryptographic construction that has the EA produce and prove, using voters' coins, the security of a common reference string (CRS) that voters subsequently can use to affix non-interactive zero-knowledge (NIZK) proofs to their ciphertexts. The EA itself uses the CRS to prove via a NIZK the tally correctness at the end. Our construction has similar performance to Helios and is practical. The privacy of our construction relies on the SXDH assumption over bilinear groups via complexity leveraging.

1 Introduction

End-to-end (E2E) verifiability has been widely identified as a critical property for the adoption of e-voting systems in real world election procedures (see e.g., [RG15] for

*Research was partly supported by ERC project CODAMODA.

†Research was supported by project FINER, Greek Secretariat of Research and Technology funded under action "ARISTEIA 1."

‡Research was performed while Zhang was a post-doc at National and Kapodistrian University of Athens, Greece.

a recent high level account). In an E2E verifiable election system, it is possible for an auditor to verify the correctness of the election tally utilizing feedback from the participants and examining the public election transcript. Naturally, E2E verifiability should be achieved without violating the privacy of the voters and any other desirable property of the election system.

Helios, [Adi08], proposed by Adida, is a widely used e-voting system that can achieve E2E verifiability and privacy (assuming it is suitably instantiated, cf. [BCP+11, BPW12]). To achieve verifiability, an important design element is the incorporation of an "audit or cast" procedure in the voting booth application that prepares the encrypted voter's choice [Ben06]. This process enables the voter to challenge her device that assists her in the preparation of her ballot. Once the voter is convinced that the device is not cheating her she can submit the enciphered vote together with a cryptographic proof (called a Non-interactive Zero-Knowledge proof or NIZK, cf. [BFM88, GS08]) that the ciphertext properly encodes the voter's choice. The inclusion of the NIZK is critical for verifiability: without it, it is possible for a malicious client to violate the encoding of the candidate choice and "stuff" the virtual ballot box with additional votes for a certain candidate of her choice (or in general invalidate the election tally).

In Helios, an auditor can verify the election tally by utilizing some feedback from the voters (specifically hashes of submitted ciphertexts that are called "smart ballot trackers") and the election transcript. The auditor will verify that ciphertexts included in the transcript match the hashes given by the voters and furthermore that all the NIZK's in the transcript are valid. If all appear to be in order the election tally can be accepted to be correct.

The above argumentation has a caveat: the verification of the NIZK's relies on the random oracle model (RO) [BR93] which basically posits that a given hash function is a random function from the perspective of the adversary. It follows that the auditor should believe the election tally as long as she believes the Election Authority (EA) has no essential understanding of the hash function[1] that gives her an advantage in breaking soundness of the underlying NIZK's. While it will be surprising to obtain a soundness attack against SHA256 based NIZK such an attack cannot be ruled out as forging SHA256-NIZK's is not a problem that has been sufficiently studied. Further, in the case of e-voting, even a single bad NIZK on a single voter ciphertext would be enough to completely corrupt the election tally and if e-voting is deployed in the large scale the EA could be subverted by a truly mighty adversary (e.g., it is believed by many that the US Government/NSA has an understanding of hash function vulnerabilities that surpasses what is publicly known).

To resolve the above concern, a system called DEMOS was put forth in [KZZ15] where it is possible to prove E2E verifiability in the "standard model", which in the terminology of that paper, means without access to an external source of randomness or the random oracle model. The main idea is to remove the task of calculating the encrypted vote from the voter client and have the EA precompute for each voter a ciphertext for each potential voter choice. The voter then casts a vote by pointing to a specific ciphertext and the EA terminates the procedure by proving all the encryptions were done correctly (actually they use commitments instead of encryptions). The system meets its objective as the EA is the sole entity that performs a Σ proof (cf. [CDS94]) with the verifier's challenge formed collectively by the voters who submit coins (that may be biased) to the election transcript.

[1] The current Helios implementation uses the SHA256 hash function for implementing NIZKs.

At first it may seem that this precomputation step is necessary to obtain this level of strong E2E verifiability. The only way to avoid precomputation from the EA, is to have the voter clients perform the encryption of the voters' choices and in such case a proof that the encryption is done correctly is needed. Such proof has to be non-interactive since the auditor will be active only after the end of the election (and in fact there may be many independent auditors that wish to check that the election was executed properly). Unfortunately NIZK's require either a "common reference string" (CRS) or the RO. In the former case, the only entity to produce the CRS is the EA and because the EA is malicious from the perspective of E2E verifiability, if she chooses the CRS, she can choose it so that the NIZK is "simulatable" and hence she may be able to produce valid looking NIZKs for an incorrect statement.

1.1 Our Results

We construct a new e-voting scheme that retains the strong E2E verifiability characteristics of [KZZ15] (but against computational adversaries) while completely obviating the need for a precomputation step by the EA. In the asymptotic sense our system has the same characteristics as Helios (while entirely removing the reliance to the RO model for security).

We achieve this via a new technique for proving the validity of ciphertexts that are submitted by the voters during ballot casting. Our proof technique may have applications beyond the e-voting domain - more comments on this below.

As mentioned above, the way to remove the precomputation is to have the voter clients produce the encrypted choices of the voter; the main technical challenge is how to prove the validity of those ciphertexts. For simplicity let us assume for now that the ciphertext ψ that is to be produced by each voter encrypts a plaintext in $\{0, 1\}$. Our construction strategy is as follows.

We will use a type of NIZK where there are two possible ways to generate the CRS, one that makes every NIZK perfectly sound and another that makes every NIZK simulatable using the trapdoor information associated to the CRS. The EA will use this dual mode CRS and will publish a CRS that is of the first type, i.e., one that makes all NIZK's perfectly sound. This is reasonable in the sense that if the EA is honest, NIZK statements produced by the voters over this CRS will be guaranteed to be valid. However, a danger comes from the fact that nobody can distinguish such CRS from the other type of CRS that is simulatable and will enable any collaborator of the EA to fake a NIZK and stuff the virtual ballot box with fake votes.

To mediate this problem we will have the EA prove that the CRS she publishes is of the first type following the same general Σ proof structure suggested in [KZZ15] (in contrast to this latter paper though, the EA will be only proving her CRS is of the first type — not that the whole election tally is valid). Subsequently any other entity (such as a voter or a trustee) can utilize the CRS of the EA (we call it the master CRS) to produce a "second layer" CRS that she can use for proving a certain statement using NIZK.

In more details our technique is as follows. The master CRS published by the EA can be seen as a public-key of an additively homomorphic encryption scheme. When the prover gets to know the statement to be shown, for instance when the voter wishes to show that a ciphertext ψ is an encryption of $\{0, 1\}$ she performs the following. She creates two strings $\mathsf{crs}_0, \mathsf{crs}_1$ that are homomorphic ciphertexts based on the master CRS of the EA. The $\mathsf{crs}_0, \mathsf{crs}_1$ strings also can be used as dual CRS's and have the

property that if they are multiplied they provide an encryption of 1. The dual property of $\mathsf{crs}_0, \mathsf{crs}_1$ is by design as follows: if crs_0 is an encryption of 0 then one can produce simulatable fake NIZK's with respect to crs_0 while if crs_0 is an encryption of 1 then one can produce only perfectly sound proofs with respect to crs_0. The same properties hold true for crs_1.

The product of the two $\mathsf{crs}_0, \mathsf{crs}_1$ is denoted by $\mathsf{crs}_{0/1}$ and we call it the second layer CRS. The prover will now show that $\mathsf{crs}_{0/1}$ is an encryption of 1 via a NIZK w.r.t. the master CRS. Then, the prover will show ψ to be an encryption of 0 with respect to crs_0 and an encryption of 1 with respect to crs_1 via two independent NIZK's on the individual CRS's. Observe that in order for the prover to accomplish this she will have to cheat in one of these two proofs; this is possible due to the fact that she has chosen the strings $\mathsf{crs}_0, \mathsf{crs}_1$ herself and she is only subject to the constraint that their product, $\mathsf{crs}_{0/1}$, is an encryption of 1 (as she has to give a NIZK proof with respect to the master CRS about this fact). This flexibility enables her to choose crs_0 or crs_1 to be an encryption of 0 and hence simulate one of the two proofs that show ψ to be an encryption of 0 or an encryption 1 (she knows the plaintext inside ψ so she has to choose crs_0 and crs_1 accordingly).

Using the above strategy, all voters can provide ciphertexts and prove them to be valid encodings of a candidate choice. It is easy to see that as long as the master CRS is of the perfectly sound type, the proofs that $\mathsf{crs}_{0/1}$ is an encryption of 1 will be perfect and hence the voters cannot stuff invalid ciphertexts in the virtual ballot box. However, as mentioned above, the EA might attempt to use a master CRS of the second type (simulatable) and then collaborate with a voter to violate integrity. To prevent that we require the EA to start a Σ proof showing that the master CRS is of the perfectly sound type. As in [KZZ15] we collect coins from the voters to form a weak random source and finally at the end have the EA publish together with the result the final move of the Σ protocol using the coins of the voters as the challenge. The auditor now is tasked with checking all the NIZK's w.r.t. the CRS's generated by the voters as well as the NIZK's provided w.r.t. the master CRS. Finally, she is also tasked with checking the Σ proof provided by the EA showing that the master CRS is of the perfectly sound type.

Given the above, one may wonder how we will be able to prove privacy. Naturally, if we are to prove privacy, we should be able to construct a type of simulation argument that enables us to plug in some instance of a hard problem into the honest voters' ciphertexts. This requires faking at least one of the NIZK's which by nature of perfect soundness we will be unable to perform. To circumvent this we utilize complexity-leveraging: in the privacy proof, our reduction will take super-polynomial time to find an execution of the adversary where the reduction can cheat the Σ protocol proof and fake it so that the master CRS is of the second type (simulatable NIZK's). In this execution our reduction will be able to fake all proofs and thus plug in any given hard problem instance. While this cannot happen in the real world (as the EA needs to run in real time so she is not expected to find such an execution) our reduction will still yield an algorithm that breaks a hard problem (it will be the symmetric external Diffie-Hellman (SXDH) problem over bilinear groups). Assuming this problem is subexponentially hard we get a contradiction and hence privacy holds.

Note that in our system we can have trustees participate (exactly as in the case of Helios) and produce the master CRS on behalf of the EA; hence we can achieve the same distribution of trust with respect to privacy as in the Helios system (this was left open in [KZZ15]). An interesting side note is that our strategy above provides an efficient way to perform a Lapidot-Shamir [LS90] type of Σ protocol: the prover performs a Σ-proof

for the CRS that, by nature, does not have to depend on the statement to be shown which can become known to the prover only in the third move of the protocol (where in our case the prover will show using a NIZK). This proof technique can be generalized and may have applications beyond e-voting.

1.2 Related Work

Our technique for structuring our proof argument is inspired by [GOS06] where correlated random strings are chosen by the prover in such a way that a verifier can check that at least one of them will yield a perfectly sound proof but she cannot distinguish which one. The prover then proceeds to issue NIZK's for each random string thus ensuring that one of them will be valid. Our NIZK proof is a modified and simplified version of the NIZK given in [GS08].

In terms of modeling privacy and verifiability we follow previous works in the area [KTV10, KTV11, BCP+11, BPW12, KZZ15] mostly using the latter reference as a basis for our formulation. Importantly, due to the virtues of our construction, we can achieve a stronger level of privacy compared to [KZZ15] that is in the simulation based sense, akin to [BPW12], as opposed to the indistinguishability-type of privacy shown in [KZZ15].

We also argue that our definition captures a level of receipt-freeness in the sense of [KZZ15]. On the other hand, we do not deal with coercion resistance; however techniques such as those of [JCJ02] are in principle compatible with our approach and our system may be augmented to incorporate them. We leave this for future work.

2 Preliminaries

2.1 Notations

Throughout this paper, we use λ as the security parameter. We use $\mathsf{negl}(\lambda)$ to denote that some function is negligible in λ. Calligraphic letters are used for sets and algorithms. A shorthand $x \leftarrow \mathcal{X}$ denotes that x is drawn uniformly from a set $\mathcal{X}$. For algorithms and distributions, the same notation $x \leftarrow \mathcal{A}$ means that the element x is sampled according to the (output) distribution. When $\mathcal{A}$ is a probabilistic algorithm, we use $\mathcal{A}(x;r)$ to denote running $\mathcal{A}$ on input x with explicate random coin r. Let $A = (A_1, A_2) \in \mathbb{G}^2$ and $B = (B_1, B_2) \in \mathbb{G}^2$ be two ElGamal ciphertexts. By $A \cdot B$, we denote the direct product $(A_1 \cdot B_1, A_2 \cdot B_2)$; by A^x we mean (A_1^x, A_2^x). We use $\mathsf{Dlog}_g(h)$ to label the discrete logarithm of h with respect to base g.

2.2 Bilinear Groups and SXDH Assumptions

Let $\mathsf{Gen}_{\mathsf{bp}}(\cdot)$ be a probabilistic polynomial time (PPT) bilinear group generator that takes as input, 1^λ and outputs the group parameters, $\sigma_{\mathsf{bp}} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where (i) $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic multiplicative groups with prime order p; (ii) $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively; (iii) $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ is a non-degenerate bilinear pairing such that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$. We assume the decisional Diffie-Hellman problem is hard in both groups, a.k.a., the *symmetric external Diffie-Hellman* (SXDH) assumption, stated in Defintion 1, holds for $\mathbb{G}_1$ and $\mathbb{G}_2$.

Definition 1 (SXDH assumption)**.** *We say SXDH assumption holds for* $\mathsf{Gen}_{\mathsf{bp}}(1^\lambda)$ *if for any PPT adversary* $\mathcal{A}$ *we have that for* $i \in \{1,2\}$*:*

$$\mathsf{Adv}_{\mathsf{sxdh}}(\mathcal{A}) := \left| \Pr\left[\begin{array}{l} \sigma_{\mathsf{bp}} \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^\lambda); \\ a, b \leftarrow \mathbb{Z}_p^* : \\ \mathcal{A}(\sigma_{\mathsf{bp}}, g_i^a, g_i^b, g_i^{ab}) = 1 \end{array} \right] - \Pr\left[\begin{array}{l} \sigma_{\mathsf{bp}} \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^\lambda); \\ a, b, c \leftarrow \mathbb{Z}_p^* : \\ \mathcal{A}(\sigma_{\mathsf{bp}}, g_i^a, g_i^b, g_i^c) = 1 \end{array} \right] \right| = \mathsf{negl}(\lambda),$$

where $\mathsf{Adv}_{\mathsf{sxdh}}(\mathcal{A})$ is the distinguishing advantage of $\mathcal{A}$.

2.3 Sigma Protocols

Let $\mathcal{L}$ be a language in **NP** and let $\mathcal{R}_{\mathcal{L}}$ be an efficiently computable binary relation for $\mathcal{L}$. For all $(x, w) \in \mathcal{R}_{\mathcal{L}}$, we call x the statement and w the witness for x in $\mathcal{L}$. In a Sigma protocol, the prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ a statement $x \in \mathcal{L}$ in a zero knowledge fashion. More specifically, Sigma protocols are 3-move public coin special honest-verifier zero knowledge proofs of knowledge with special soundness. In the first step, the prover publishes a commitment message, denoted as $\Sigma_{(1)}$. In the second step, the verifier gives a challenge message, denoted as ch. In the third step, the prover outputs a response message, denoted as $\Sigma_{(2)}$. At the end, the transcript $(\Sigma_{(1)}, \mathsf{ch}, \Sigma_{(2)})$ is publicly verifiable with respect to $(x, \mathcal{L})$.

In this work, we employ the Schnorr's identity protocol [Sch89] to show the knowledge of discrete logarithm. For instance, $\Sigma^{\mathsf{dlog}}\{(s) : h = g^s\}$ stands for a Sigma protocol for the knowledge of secret s satisfying the equation $h = g^s$. We also use the DDH tuple proof [Cha90] to show the correctness of election parameters. In particular, by $\Sigma^{\mathsf{ddh}}\{(x) : C = A^x \wedge D = B^x\}$, we mean a Sigma protocol showing the DDH tuple relation of (A, B, C, D).

2.4 Non-interactive Zero Knowledge Proofs.

A *non-interactive proof system* $\Gamma = (\mathsf{Gen}_{\mathsf{crs}}, \mathsf{Sim}_{\mathsf{crs}}, \mathsf{Prov}, \mathsf{Vrfy}, \mathsf{Sim})$ for group languages consists of the following PPT algorithms:

- $\mathsf{Gen}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$ is a CRS generation algorithm that takes as input the bilinear group parameter $\sigma_{bp} \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^\lambda)$ and outputs a common reference string crs.
- $\mathsf{Sim}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$ is a CRS simulator that takes as input the bilinear group parameter $\sigma_{bp} \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^\lambda)$ and outputs a simulated common reference string crs^* together with a simulation trapdoor td.
- $\mathsf{Prov}(\mathsf{crs}; x; w)$ is the prover algorithm that takes as input the common reference string crs, the statement x and the witness w, and outputs a proof π.
- $\mathsf{Vrfy}(\mathsf{crs}; x; \pi)$ is the verifier algorithm that takes as input the common reference string crs, the statement x and the proof π, and outputs 1 if the proof is acceptable and 0 otherwise.
- $\mathsf{Sim}(\mathsf{crs}^*; x; \mathsf{td})$ is the proof simulator that takes as input the simulated crs^*, the statement x and the simulation trapdoor td, and outputs a simulated proof π^*.

Definition 2 (NIZK)**.** *We say that* $\Gamma = (\mathsf{Gen}_{\mathsf{crs}}, \mathsf{Sim}_{\mathsf{crs}}, \mathsf{Prov}, \mathsf{Vrfy}, \mathsf{Sim})$ *is a* non-interactive zero knowledge *(NIZK) proof system for group language* $\mathcal{L}$ *if it has completeness, soundness, and zero knowledge properties described below.*

1. Perfect completeness:

$$\Pr\left[\begin{array}{l}\sigma_{\mathsf{bp}} \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^\lambda); \mathsf{crs} \leftarrow \mathsf{Gen}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}); (x,w) \leftarrow \mathcal{A}(\mathsf{crs}); \\ \pi \leftarrow \mathsf{Prov}(\mathsf{crs}; x; w) : \mathsf{Vrfy}(\mathsf{crs}; x; \pi) = 1 \vee (x,w) \notin \mathcal{R}_{\mathcal{L}}\end{array}\right] = 1$$

2. Perfect soundness: *for all adversaries* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{l}\sigma_{\mathsf{bp}} \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^\lambda); \mathsf{crs} \leftarrow \mathsf{Gen}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}); \\ (x,\pi) \leftarrow \mathcal{A}(\mathsf{crs}) : \mathsf{Vrfy}(\mathsf{crs}; x; \pi) = 1 \wedge x \notin \mathcal{L}\end{array}\right] = 0$$

3. Computational zero-knowledge: *there exists a pair of probabilistic polynomial time simulators* $(\mathsf{Sim}_{\mathsf{crs}}, \mathsf{Sim})$ *such that for all non-uniform polynomial time adversaries* $\mathcal{A}$*,*

$$\left|\Pr\left[\begin{array}{l}\sigma_{\mathsf{bp}} \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^\lambda); \\ \mathsf{crs} \leftarrow \mathsf{Gen}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}) : \\ \mathcal{A}^{\mathsf{Prov}(\mathsf{crs};\cdot;\cdot)}(\mathsf{crs}) = 1\end{array}\right] - \Pr\left[\begin{array}{l}\sigma_{\mathsf{bp}} \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^\lambda); \\ (\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{Sim}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}) : \\ \mathcal{A}^{\mathsf{Sim}^*(\mathsf{crs}^*,\mathsf{td},\cdot,\cdot)}(\mathsf{crs}^*) = 1\end{array}\right]\right| = \mathsf{negl}(\lambda),$$

where $\mathsf{Sim}^*(\mathsf{crs}^*, \mathsf{td}, x, w) = \mathsf{Sim}(\mathsf{crs}^*, x, \mathsf{td})$ *for* $(x,w) \in \mathcal{R}_{\mathcal{L}}$ *and both oracles output* $\perp$ *if* $(x,w) \notin \mathcal{R}_{\mathcal{L}}$*.*

3 E-voting Security Framework

3.1 Syntax

An e-voting system is an interactive protocol Π among an election authority EA, a bulletin board BB, a set of voters $\mathcal{V} = \{V_1, \ldots, V_n\}$ (who may utilize a *voter supporting device* (VSD) to vote), and a set of trustees $\mathcal{T} = \{T_1, \ldots, T_t\}$. Let $\mathcal{O} = \{\mathsf{opt}_1, ..., \mathsf{opt}_m\}$ be the set of election options. We denote by $\mathcal{U} \subseteq 2^{\mathcal{O}}$ the collection of subsets of options that the voters are allowed to choose to vote. We denote the option selection of voter V_ℓ as $\mathcal{U}_\ell \in \mathcal{U}$, which is a subset of the options.

Let $\vec{\mathcal{U}}$ be the set of vectors of option selections of arbitrary length. Let $F : \vec{\mathcal{U}} \mapsto (\mathbb{Z}_+)^m$ be the election evaluation function such that $F(\mathcal{U}_1, \ldots, \mathcal{U}_n)$ is equal to an m-vector whose i-th location is equal to the number of times opt_i was chosen in the option selections $\mathcal{U}_1, \ldots, \mathcal{U}_n$.

Similar as [KZZ15], we consider an e-voting system Π as a quintuple of algorithms and protocols that are denoted by (**Setup**, **Cast**, **Tally**, **Result**, **Verify**) as follows:

- **Setup** is a protocol executed among the trustees $\mathcal{T}$, the election authority EA, and the bulletin board BB. During the protocol, the election public parameters including $\mathcal{V}, \mathcal{O}, \mathcal{U}$ are generated and published on BB. The voters' credentials $(s_1, \ldots, s_n)$ are generated and distributed to the voters. After the protocol execution, each trustee T_i obtains a private state st_i.

- **Cast** is a protocol executed between the voter V_ℓ (who operates a VSD) and BB. In the protocol, V_ℓ posts her ballot $\mathbf{blt}_\ell$ to BB and obtains a receipt rec_ℓ.

- **Tally** is a protocol executed among $T_1, \ldots, T_t$, EA and BB. In the protocol, each T_i sends her partial tally data $\mathbf{ta}_i$ to the EA, and the EA will post the received data to BB after all T_i complete their **Tally** protocols.

The ideal election process for privacy $\mathcal{F}_{\mathsf{priv}}^{m,n}$.

- Upon receiving $(sid, \mathsf{init}, \mathcal{O}, \mathcal{V}, \mathcal{U})$ from EA, it parses $\mathcal{O}$ as options $\{\mathsf{opt}_1, ..., \mathsf{opt}_m\}$, $\mathcal{V}$ as voters $\{V_1, ..., V_n\}$, and $\mathcal{U}$ voting option selections $\mathcal{U} \subseteq 2^{\mathcal{O}}$. It sets the election status to 'vote' and initiliazes a list records as empty. Finally, it sends $(sid, \mathsf{vote}, \mathcal{O}, \mathcal{V}, \mathcal{U})$ to the adversary $\mathcal{S}$.
- Upon receiving $(sid, \mathsf{cast}, \mathcal{U}_\ell)$ from V_ℓ, if the election status is 'vote' and $\mathcal{U}_\ell \in \mathcal{U}$, then it sends $(sid, \mathsf{cast}, V_\ell)$ to $\mathcal{S}$.
- Upon receiving $(sid, \mathsf{cast}, V_\ell)$ from $\mathcal{S}$, if the election status is 'vote' and $(sid, \mathsf{cast}, \mathcal{U}_\ell)$ was sent before by V_ℓ, it adds $(V_\ell, \mathcal{U}_\ell)$ to records.
- Upon receiving (sid, tally) from $\mathcal{S}$, if the election status is 'vote', it sets the election status to 'tally' and computes the election result $\tau \leftarrow F(\langle \mathcal{U}_\ell \rangle_{(V_\ell, \mathcal{U}_\ell) \in \mathsf{records}})$. Finally, it sends τ to $\mathcal{S}$.

Figure 1: The ideal process $\mathcal{F}_{\mathsf{priv}}^{m,n}$ interacting with the ideal world adversary $\mathcal{S}$.

- **Result** is an algorithm that takes as input the election transcript info on BB, and outputs the election result or returns $\perp$ in case such result is undefined.
- **Verify** is an algorithm that takes as input the election transcript info on BB and an auxiliary information aux, and outputs 0 or 1. aux can be either a voter's receipt, rec_ℓ or a trustee's private state st_i.

3.2 Simulation-based Privacy

We model privacy as indistinguishability between a real-world experiment and an ideal world experiment.

In the ideal experiment, we consider an ideal process $\mathcal{F}_{\mathsf{priv}}^{m,n}$ defined in Figure 1 that captures the essential aspects of the election functionality from a privacy perspective (we stress that this is not a full ideal functionality as it is not intended to capture correctness or verifiability which we model separately). All the voters $V_1, \ldots, V_n$, are modeled as dummy parties that simply forward the inputs they receive from the environment $\mathcal{Z}$ to the ideal process $\mathcal{F}_{\mathsf{priv}}^{m,n,t}$. Note that the environment $\mathcal{Z}$ can schedule all the election entities arbitrarily. The ideal world adversary $\mathcal{S}$ that is active in the experiment interacts with $\mathcal{F}_{\mathsf{priv}}^{m,n}$ and provides output to $\mathcal{Z}$ which makes a final decision outputing a bit. Note that the interaction between $\mathcal{Z}$ and $\mathcal{S}$ is restricted in this way (in the spirit of [Can98]) since in our setting it is impossible to achieve stronger notions of simulation-based security (such as universal composition, [Can01]). We denote the output of the environment in the ideal experiment by $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{priv}}^{m,n}, \mathcal{S}, \mathcal{Z}}(\lambda)$.

In the real world experiment, the entities $\mathcal{T} = T_1, \ldots, T_t$, $\mathcal{V} = V_1, \ldots, V_n$, EA, BB participate in the e-voting system $\Pi = (\mathbf{Setup}, \mathbf{Cast}, \mathbf{Tally}, \mathbf{Result}, \mathbf{Verify})$ in the presence of an adversary $\mathcal{A}$ who has corrupted up to k voters and $t-1$ trustees. The voters and the trustees run the protocol on command by the environment $\mathcal{Z}$. We denote the output of the environment in the real world experiment by $\mathsf{REAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\lambda)$. The objective of the adversary is to obtain sufficient information about the honest voters' option selection so that, in collaboration with the environment, is able to distinguish the real from the ideal world execution.

We say that the e-voting system is private if for all real-world adversaries $\mathcal{A}$ there is a simulator $\mathcal{S}$ so that it is impossible for any environment $\mathcal{Z}$ to distinguish between the real and ideal world experiment. Formally, we define it as follows.

Definition 3. *Let $n, m, t, k \in \mathbb{N}$ with $k \leq n$ and let $\Pi = (\mathbf{Setup}, \mathbf{Cast}, \mathbf{Tally}, \mathbf{Result}, \mathbf{Verify})$ be an e-voting system with t trustees and n voters. We say that Π is k-private if for every PPT adversary $\mathcal{A}$ controlling up to k voters and up to $t-1$ trustees, there is an adversary $\mathcal{S}$ in the ideal world experiment, such that for every environment $\mathcal{Z}$ the random variables $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{priv}}^{m,n},\mathcal{S},\mathcal{Z}}(\lambda)$ and $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$ are computationally indistinguishable.*

3.3 E2E Verifiability

We would like to extend the E2E verifiability definition in [KZZ15] by adding VSD and trustees. Similarly, we use the metric $\mathrm{d}_1(\cdot,\cdot)$ derived by the 1-norm, $\|\cdot\|_1$ scaled to half, i.e.,

$$\begin{array}{rccl} \mathrm{d}_1: & \mathbb{Z}_+^m \times \mathbb{Z}_+^m & \longrightarrow & \mathbb{R} \\ & (w, w') & \longmapsto & \mathrm{d}_1(w, w') = \frac{1}{2} \cdot \|w - w'\|_1 = \frac{1}{2} \cdot \sum_{i=1}^{n} |w_i - w'_i| \end{array}$$

to measure the adversarial success rate with respect to the amount of tally deviation d and the number of voters that perform audit θ. The adversary starts by selecting the voter, options and trustee identities for given parameters n, m, k. It also specifies the set $\mathcal{U}$ of the allowed ways to vote. The adversary now fully controls all the trustees and the EA. The adversary manages the **Cast** protocol executions, playing the role of the VSD. For each voter, the adversary may choose to corrupt it or to allow the challenger to play on its behalf. In the second case, the adversary provides the option selection that the honest voter will use in the **Cast** protocol. The adversary finally posts the election transcript to the BB.

As in [KZZ15], we consider a *vote extractor* algorithm $\mathcal{E}$ (not necessarily running in polynomial-time) that explains the election transcript from the dishonest voters' aspect. The adversary will win the game provided that there is a subset of θ honest voters that audit the result successfully but the deviation of the tally is bigger than d; the adversary will also win in case the vote extractor fails to produce the option selection of the dishonest voters but still, θ honest voters verify correctly. The attack game is specified in detail in Figure 2.

Definition 4 (E2E-Verifiability)**.** *Let $0 < \epsilon < 1$ and $m, n, t, d, \theta \in \mathbb{N}$ with $\theta \leq n$ and Π an e-voting protocol with n voters and t trustees. Π achieves* E2E verifiability *with error ϵ, w.r.t. the election function F, a number of θ honest successfull voters and tally deviation d if there exists a (not necessarily polynomial-time) vote extractor $\mathcal{E}$ such that for any PPT adversary $\mathcal{A}$ it holds that*

$$\Pr[G_{\mathsf{e2e\text{-}ver}}^{\mathcal{A},\mathcal{E},d,\theta}(1^\lambda, m, n, t) = 1] < \epsilon.$$

In plain words, Definition 4 implies that in an E2E verifiable e-voting system, if the number of voters that verify the result is θ then any adversary that attempts to manipulate the result by a shift of d according to the metric $\mathrm{d}_1(\cdot,\cdot)$ will get caught with probability more than $1 - \epsilon$.

E2E Verifiability Game $G^{\mathcal{A},\mathcal{E},d,\theta}_{\mathsf{e2e\text{-}ver}}(1^\lambda, m, n, t)$

1. $\mathcal{A}$ chooses $\mathcal{O} = \{\mathsf{opt}_1, ..., \mathsf{opt}_m\}$, $\mathcal{V} = \{V_1, ..., V_n\}$, $\mathcal{T} = \{T_1, ..., T_t\}$, and $\mathcal{U} \subseteq 2^{\mathcal{O}}$. It sends them to $\mathcal{C}$ along with some public election parameters and voter credentials $\{s_\ell\}_{\ell\in[n]}$ parameterized by security parameter λ. Throughout the game, $\mathcal{C}$ plays the role of the BB.
2. $\mathcal{A}$ and $\mathcal{C}$ engage 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 $\mathcal{C}$ operate on V_ℓ's behalf, in which case $\mathcal{A}$ provides an option selection $\mathcal{U}_\ell$ to $\mathcal{C}$. Then, $\mathcal{C}$ engages in the **Cast** protocol with the adversary $\mathcal{A}$, so that $\mathcal{A}$ plays the roles of EA and VSD. 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 info to the BB..

The game returns a bit which is 1 if and only if the following conditions hold true:

1. $|\tilde{\mathcal{V}}| \geq \theta$ (i.e., at least θ honest voters terminated).
2. $\forall \ell \in [n]$: if $V_\ell \in \tilde{\mathcal{V}}$ then $\mathbf{Verify}(\mathsf{info}, \mathsf{rec}_\ell) = 1$.

and either one of the following two :

3.a. if $\perp \neq \langle \mathcal{U}_\ell \rangle_{V_\ell \in \mathcal{V}\setminus\tilde{\mathcal{V}}} \leftarrow \mathcal{E}(\mathsf{info}, \{\mathsf{rec}_\ell\}_{V_\ell\in\tilde{V}})$ then

$\mathrm{d}_1\big(\mathbf{Result}(\mathsf{info}), F(\mathcal{U}_1, \ldots, \mathcal{U}_n)\big) \geq d$,

3.b. $\perp \leftarrow \mathcal{E}(\mathsf{info}, \{\mathsf{rec}_\ell\}_{V_\ell\in\tilde{\mathcal{V}}})$.

Figure 2: The E2E Verifiability Game between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ using the vote extractor $\mathcal{E}$.

4 Building Blocks

4.1 A NIZK for DDH Tuple

In this section, we construct a NIZK proof that allows the prover to convince the verifier that $(A, B, C, D) \in (\mathbb{G}_1)^4$ is a DDH tuple. Namely, the prover shows that he knows $s \in \mathbb{Z}_p$ such that $C = A^s \wedge D = B^s$. The same proof works analogously for DDH tuples in $(\mathbb{G}_2)^4$, and such a NIZK proof with respect to crs is denoted as $\mathsf{NIZK}\{\mathsf{crs}; (s) : C = A^s \wedge D = B^s\}$.

Our proof can be seen as a simplification of the well-known Groth-Sahai (GS) proof system [GS08]. Similar to [GOS06], we use additively homomorphic public key cryptosystem, lifted ElGamal, for the commitment scheme. Given an lifted ElGamal ciphertext $u \in (\mathbb{G}_2)^2$ under public key pk, the commitment of $m \in \mathbb{Z}_p$ with randomness $r \in \mathbb{Z}_p$ under commitment key $\mathsf{ck} = (\mathsf{pk}, u)$ is defined as

$$\boxed{\mathsf{Com}_{\mathsf{ck}}(m; r) := u^m \cdot \mathsf{Enc}_{\mathsf{pk}}(0; r)}$$

. It is easy to see that:

1. If u is an encryption of a non-zero value x, then the commitment $\mathsf{Com}_{\mathsf{ck}}(m; r)$ is an encryption of xm and it is perfectly binding.
2. If u is an encryption of 0, then for very m the commitment $\mathsf{Com}_{\mathsf{ck}}(m; r)$ is an encryption of 0 and it is perfectly hiding.

The CRS of our NIZK consists of the bilinear group parameter, σ_{bp} and the commitment key, $\mathsf{ck} := (\mathsf{pk}, u)$. The CRS is *perfectly sound* when the perfectly binding commitment key is used, while it is *perfectly simulatable* when the perfectly hiding commitment key is used. Formally, the NIZK proof system Γ^{ddh} consists of the following PPT algorithms:

- $\mathsf{Gen}^{\mathsf{ddh}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$:
  - Pick $\alpha_1, \alpha_2 \leftarrow \mathbb{Z}_p^*$;
  - Set $h_2 := g_2^{\alpha_1}$ and $u := (u_1, u_2) := (g_2^{\alpha_2}, g_2 h_2^{\alpha_2})$;
  - Set $\mathsf{ck} := (h_2, u)$;
  - Output $\mathsf{crs} := (\sigma_{\mathsf{bp}}, \mathsf{ck})$;
- $\mathsf{Sim}^{\mathsf{ddh}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$:
  - Pick $\alpha_1, \alpha_2 \leftarrow \mathbb{Z}_p^*$;
  - Set $h_2 := g_2^{\alpha_1}$ and $u = (u_1, u_2) := (g_2^{\alpha_2}, h_2^{\alpha_2})$;
  - Set $\mathsf{ck} := (h_2, u)$;
  - Output $\mathsf{crs}^* := (\sigma_{\mathsf{bp}}, \mathsf{ck})$ and $\mathsf{td} := \alpha_2$;
- $\mathsf{Prov}^{\mathsf{ddh}}(\mathsf{crs}; (A, B, C, D); s)$:
  - Pick $r \leftarrow \mathbb{Z}_p$;
  - Set $c := (c_1, c_2) = \mathsf{Com}_{\mathsf{ck}}(s; r) := (u_1^s g_2^r, u_2^s h_2^r)$, $\pi_1 := A^r$, and $\pi_2 := B^r$;
  - Output $\pi := (c, \pi_1, \pi_2)$;
- $\mathsf{Vrfy}^{\mathsf{ddh}}(\mathsf{crs}; (A, B, C, D); \pi)$:
  - Output 1 if and only if the following hold:
    - $e(C, u_1) \cdot e(\pi_1, g_2) = e(A, c_1)$;
    - $e(C, u_2) \cdot e(\pi_1, h_2) = e(A, c_2)$;
    - $e(D, u_1) \cdot e(\pi_2, g_2) = e(B, c_1)$;
    - $e(D, u_2) \cdot e(\pi_2, h_2) = e(B, c_2)$;
- $\mathsf{Sim}^{\mathsf{ddh}}(\mathsf{crs}^*; (A, B, C, D); \mathsf{td})$:
  - Pick $r \leftarrow \mathbb{Z}_p$;
  - Set $c^* = (c_1, c_2) := (g_2^r, h_2^r)$, $\pi_1^* = A^r C^{-\alpha_2}$, and $\pi_2^* = B^r D^{-\alpha_2}$;
  - Output $\pi^* := (c^*, \pi_1^*, \pi_2^*)$;

Clearly, the simulated CRS is computationally indistinguishable from the real CRS based on the IND-CPA security of the underlying ElGamal cryptosystem. We state the following theorem without providing the proof since it can be directly derive from the generic GS proof for the SXDH instantiation in [GS08].

Theorem 1. *The protocol Γ^{ddh} is a NIZK proof system for the language*

$$\mathcal{L}^{\mathsf{ddh}} = \left\{ (A, B, C, D) \in (\mathbb{G}_1)^4 \mid \exists s : C = A^s \wedge D = B^s \right\},$$

i.e. (A, B, C, D) is a DDH tuple. The NIZK proof has perfect completeness, perfect soundness and computational zero-knowledge under the SXDH assumption.

4.2 NIZK OR Composition

In our work, OR composition of the NIZK proofs is often needed, e.g., to show a lifted ElGamal ciphertext in $(\mathbb{G}_1)^2$ (resp. $(\mathbb{G}_2)^2$) is an encryption of 0 or 1. To achieve this, we adopt the *correlated key generation* technique from [GOS06][2]. The intuition is to use two tiers of NIZK proofs, where the CRS for the first tier NIZK is given as the master CRS. To prove an OR composition of statements such as $x_1 \vee \ldots \vee x_n$, the prover first generates n second tier CRS's, $\mathsf{crs}_1, \ldots, \mathsf{crs}_n$ and uses the master CRS to show that at least one of them is a perfectly sound CRS; the prover then uses the second tier CRS crs_i to prove the statement x_i for $i \in [n]$. Since the prover is able to generate $n-1$ perfectly simulatable CRS's with trapdoors, he can simulate any $n-1$ statements. On the other hand, at least one of the crs_i is perfectly sound, so at least one of the statement x_i is valid. The ZK property directly follows the fact that it is computationally hard to distinguish which CRS is perfectly sound.

More specifically, the prover gives n lifted ElGamal ciphertexts as the n second tier CRS, and shows the product of them is an encryption of 1 using the DDH tuple NIZK described in Section 4.1. Therefore, we can ensure that at least one of the CRS encrypts an non-zero value. In the following, we describe two special cases of OR compostion that we apply in our e-voting system.

4.2.1 Proving that a ciphertext encrypts 0 or 1

We describe the NIZK proof system $\Gamma^{0/1}$ for the ciphertext $c = \mathsf{Enc}_{\mathsf{pk}}(b; r) \in (\mathbb{G}_1)^2$ encrypts 0 or 1, i.e., $b \in \{0, 1\}$.

- $\mathsf{Gen}^{0/1}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$:
  - Use $\mathbb{G}_1$ variant of $\mathsf{Gen}^{\mathsf{ddh}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$ to produce a master CRS $\mathsf{crs}_{\mathsf{m}}$ in $\mathbb{G}_1$;
- $\mathsf{Sim}^{0/1}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$:
  - Use $\mathbb{G}_1$ variant of $\mathsf{Sim}^{\mathsf{ddh}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$ to produce a simulated CRS crs^*_m in $\mathbb{G}_1$ and a trapdoor td;
- $\mathsf{Prov}^{0/1}(\mathsf{crs}_{\mathsf{m}}; (\mathsf{pk} := (g_1, f_1), c); (b, r))$:
  - Pick $\alpha_1, \alpha_2, \alpha_3 \leftarrow \mathbb{Z}_p$;
  - Set $h_2 := g_2^{\alpha_1}$, $u^{(b)} := (u_1^{(b)}, u_2^{(b)}) = (g_2^{\alpha_2}, g_2 h_2^{\alpha_2})$, and $u^{(1-b)} := (u_1^{(1-b)}, u_2^{(1-b)}) = (g_2^{\alpha_3}, h_2^{\alpha_3})$;
  - Set $\mathsf{ck}^{(b)} := (h_2, u^{(b)})$ and $\mathsf{ck}^{(1-b)} := (h_2, u^{(1-b)})$;
  - Define $\mathsf{crs}^{(b)} := (\sigma_{\mathsf{bp}}, \mathsf{ck}^{(b)})$ and $\mathsf{crs}^{(1-b)} := (\sigma_{\mathsf{bp}}, \mathsf{ck}^{(1-b)})$;
  - Set $(u_1, u_2) = u^{(b)} \cdot u^{(1-b)} \in (\mathbb{G}_2)^2$;
  - Compute $\pi_{\mathsf{crs}} \leftarrow \mathsf{Prov}^{\mathsf{ddh}}(\mathsf{crs}_{\mathsf{m}}; (g_2, h_2, u_1, u_2/g_2); \alpha_2 + \alpha_3)$;
  - Set $\pi^{(b)} \leftarrow \mathsf{Prov}^{\mathsf{ddh}}(\mathsf{crs}^{(b)}; (g_1, f_1, c_1, c_2/g_1^b); r)$ and $\pi^{(1-b)} \leftarrow \mathsf{Sim}^{\mathsf{ddh}}(\mathsf{crs}^{(1-b)}; (g_1, f_1, c_1, c_2/g_1^{1-b}); \alpha_3)$;

[2] We refer interested readers to [Ràf15] for more general NIZK composition via correlated key generation.

– Output $\pi := (\mathsf{crs}^{(0)}, \mathsf{crs}^{(1)}, \pi_{\mathsf{crs}}, \pi^{(0)}, \pi^{(1)})$;

- $\mathsf{Vrfy}^{0/1}(\mathsf{crs_m}; (\mathsf{pk} := (g_1, f_1), c); \pi)$:
  - Output 1 if and only if the following verifies.
    - $\mathsf{Vrfy}^{\mathsf{ddh}}(\mathsf{crs_m}; (g_2, h_2, u_1, u_2/g_2); \pi_{\mathsf{crs}})$=1;
    - $\mathsf{Vrfy}^{\mathsf{ddh}}(\mathsf{crs}^{(0)}; (g_1, f_1, c_1, c_2); \pi^{(0)}) = 1$;
    - $\mathsf{Vrfy}^{\mathsf{ddh}}(\mathsf{crs}^{(1)}; (g_1, f_1, c_1, c_2/g_2); \pi^{(1)}) = 1$;
- $\mathsf{Sim}^{0/1}(\mathsf{crs}^*_m; (\mathsf{pk} := (g_1, f_1), c); \mathsf{td})$:
  - Pick $\alpha_1, \alpha_2, \alpha_3 \leftarrow \mathbb{Z}_p$;
  - Set $h_2 := g_2^{\alpha_1}$, $u^{(0)} = (u_1^{(0)}, u_2^{(0)}) := (g_2^{\alpha_2}, h_2^{\alpha_2})$, and $u^{(1)} := (u_1^{(1)}, u_2^{(1)}) = (g_2^{\alpha_3}, h_2^{\alpha_3})$;
  - Set $\mathsf{ck}^{(0)} := (h_2, u^{(0)})$ and $\mathsf{ck}^{(1)} := (h_2, u^{(1)})$;
  - Define $\mathsf{crs}^{(0)} := (\sigma_{\mathsf{bp}}, \mathsf{ck}^{(0)})$ and $\mathsf{crs}^{(1)} := (\sigma_{\mathsf{bp}}, \mathsf{ck}^{(1)})$;
  - Set $(u_1, u_2) = u^{(0)} \cdot u^{(1)} \in (\mathbb{G}_2)^2$;
  - Compute $\pi_{\mathsf{crs}} \leftarrow \mathsf{Sim}^{\mathsf{ddh}}(\mathsf{crs}^*_m; (g_2, h_2, u_1, u_2/g_2); \mathsf{td})$;
  - Set $\pi^{(0)} \leftarrow \mathsf{Sim}^{\mathsf{ddh}}(\mathsf{crs}^{(0)}; (g_1, f_1, c_1, c_2); \alpha_2)$ and $\pi^{(1)} \leftarrow \mathsf{Sim}^{\mathsf{ddh}}(\mathsf{crs}^{(1)}; (g_1, f_1, c_1, c_2/g_1); \alpha_3)$;
  - Output $\pi^* := (\mathsf{crs}^{(0)}, \mathsf{crs}^{(1)}, \pi_{\mathsf{crs}}, \pi^{(0)}, \pi^{(1)})$;

Theorem 2. *The protocol $\Gamma^{0/1}$ is a NIZK proof system for c encrypts 0 or 1. The NIZK proof has perfect completeness, perfect soundness and computational zero-knowledge under the SXDH assumption.*

Proof. **Perfect completeness.** It directly follows from the completeness and simulatability of the underlying NIZK proof Γ^{ddh}.

Perfect soundness. The prover generates two CRSs, $\mathsf{crs}^{(0)}$ and $\mathsf{crs}^{(1)}$, and uses Γ^{ddh} to show that the product of them is lifted ElGamal encryption of 1. Since Γ^{ddh} is perfect sound, it is sure that at least one CRS encrypts to a non-zero value. By simultaneously showing the given ciphertext c is encryption of 0 and 1 with respect to $\mathsf{crs}^{(0)}$ and $\mathsf{crs}^{(1)}$, we guarantee that c encrypts either 0 or 1.

Computational zero-knowledge. It is straightforward that if the SXDH assumption holds, then $\mathsf{crs}^{(0)}$ and $\mathsf{crs}^{(1)}$ are computationally indistinguishable (hence DDH is hard for $\mathbb{G}_2$) and the simulated CRS crs^*_m is computationally indistinguishable from the real one crs_m. Moreover, the Γ^{ddh} is computationally zero-knowledge, so all the simulated sub-proofs are indistinguishable from the real ones. Therefore, π^* is computationally indistinguishable from π. ■

4.2.2 Proving that a ciphertext encrypts a value between min and max

Observe that this case is a generalization of $\Gamma^{0/1}$, where we set min = 0 and max = 1. The description follows the lines of $\Gamma^{0/1}$ where now we generate $\max - \min + 1$ CRSs denoted by $\mathsf{crs}^{(\min)}, \ldots, \mathsf{crs}^{(\max)}$. For $j \in [\min, \max]$, $\mathsf{crs}^{(j)}$ contains σ_{bp} and the commitment key $\mathsf{ck}^{(j)}$, which in turn consists of a random element $h_2 \in \mathbb{G}_2$ and an ElGamal encryption of j, $u^{(j)}$.

- $\mathcal{P} \to \mathcal{V}$: $\mathsf{crs}^{\mathcal{L}}$ and $\Sigma^{\mathsf{crs}}_{(1)}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$
- $\mathcal{V} \to \mathcal{P}$: ch
- $\mathcal{P} \to \mathcal{V}$: x, $\pi \leftarrow \mathsf{NIZK}\left\{\mathsf{crs}^{\mathcal{L}}; (w) : (x, w) \in \mathcal{R}\right\}$ and $\Sigma^{\mathsf{crs}}_{(2)}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$

Figure 3: The message structure of the composed 3-move ZK for $\mathcal{L}$.

4.3 Lapidot-Shamir Revisited

Kiayias, Zacharias and Zhang [KZZ15] proposed a technique that allows the EA to prove the validity of some cryptographic elements on the BB using Sigma protocols. In particular, the EA posts the commitment messages of Sigma protocols on the BB before the election starts; During the election, the verifier's challenge is jointly contributed by all the voters (1 bit per voter); After the election ends, the EA then completes the Sigma protocols by posting the corresponding response messages on the BB. However, this technique has its limitations, namely, the statement to be proven must be fixed before the election starts. In order to prove a statement that is generated during or after the election, we need a generic 3-move ZK protocol whose commitment message is independent of the statement, such as the Lapidot-Shamir protocol[3] [LS90]. Unfortunately, one has to convert the original language to Hamiltonian cycle in order to use the Lapidot-Shamir protocol, so it is very inefficient in practice.

To resolve this issue, we propose a new Lapidot-Shamir like 3-move ZK framework where the prover's first move does not depend on the statement to be proven. The idea is to combine a 3-move public coin honest-verifier zero-knowledge protocol with a perfectly sound NIZK proof. For notation simplicity, we will use Sigma protocol notation for such 3-move public coin honest-verifier zero-knowledge protocols, but we emphasize that the special soundness and special ZK properties are not necessary for our composition. Let $\Gamma^{\mathcal{L}}$ be a perfectly sound NIZK proof system for some NP language $\mathcal{L}$, and let $\Sigma^{\mathsf{crs}}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$ be a Sigma protocol to show the given $\mathsf{crs}^{\mathcal{L}}$ is a perfectly sound CRS. The message structure of the composed 3-move ZK protocol between the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ is depicted in Figure 3. In the first move, the prover $\mathcal{P}$ generates a NIZK CRS $\mathsf{crs}^{\mathcal{L}}$ and sends it to the verifier $\mathcal{V}$ together with the commitment message of $\Sigma^{\mathsf{crs}}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$.In the second move the verifier $\mathcal{V}$ gives the challenge message ch. In the third move, the prover $\mathcal{V}$ fixes the statement $x \in \mathcal{L}$ and computes the NIZK proof for x, $\pi \leftarrow \mathsf{NIZK}\left\{\mathsf{crs}^{\mathcal{L}}; (w) : (x, w) \in \mathcal{R}\right\}$. $\mathcal{P}$ then sends to $\mathcal{V}$ the statement x, the NIZK proof π, and the response message of $\Sigma^{\mathsf{crs}}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$. $\mathcal{V}$ accepts the proof if (i) $(\Sigma^{\mathsf{crs}}_{(1)}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}, \mathsf{ch}, \Sigma^{\mathsf{crs}}_{(2)}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\})$ is a valid Sigma protocol transcript and (ii) $\mathsf{Vrfy}^{\mathcal{L}}(\mathsf{crs}^{\mathcal{L}}; x; \pi) = 1$.

Theorem 3. *Let $\Gamma^{\mathcal{L}}$ be a perfectly complete, perfectly sound, and computationally zero-knowledge NIZK proof system for language $\mathcal{L}$, and let $\Sigma^{\mathsf{crs}}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$ be a 3-move public coin honest verifier zero-knowledge protocol with perfectly completeness, statistical soundness, and computational zero-knowledge. The above composed 3-move*

[3]NB: Technically, the size of the commitment message of the Lapidot-Shamir protocol still depends on the statement.

public coin honest verifier zero-knowledge protocol for language $\mathcal{L}$ is perfectly complete, statistically sound, and computationally zero-knowledge.

Proof. **Perfect completeness.** It directly follows from the perfect completeness properties of both $\Gamma^{\mathcal{L}}$ and $\Sigma^{\mathsf{crs}}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$.

Statistical soundness. Since $\Sigma^{\mathsf{crs}}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(1^{\lambda}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$ is statistically sound, $\mathsf{crs}^{\mathcal{L}}$ is a perfectly sound CRS with over-whelming probability. When $\mathsf{crs}^{\mathcal{L}}$ is a perfectly sound CRS, no adversary can produce a fake π^* to make the verifier accept an invalid $x^* \notin \mathcal{L}$ such that $\mathsf{Vrfy}^{\mathcal{L}}(\mathsf{crs}^{\mathcal{L}}; x^*; \pi^*) = 1$. Hence, the composed ZK is statistically sound.

Computational zero-knowledge. The simulatable CRS crs^* generated by $\mathsf{Sim}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}})$ is computationally indistinguishable from a perfectly sound CRS. From the computationally zero-knowledge properties of both $\Gamma^{\mathcal{L}}$ and $\Sigma^{\mathsf{crs}}\left\{(r) : \mathsf{Gen}^{\mathcal{L}}_{\mathsf{crs}}(\sigma_{\mathsf{bp}}; r) = \mathsf{crs}^{\mathcal{L}}\right\}$, it is easy to see that the simulated composed ZK proof is computationally indistinguishable from a real one. ■

5 System Design

5.1 System Overview

Our system is a single-server web-based system. In an election, the election server mainly plays the roles of the EA and BB but may aid the other parties. All the other parties are realized by Javascripts running at the client side. We assume there is a secure channel between the election server and each trustee, say, realized by HTTPS. The system uses homomorphic tally and currently supports x-out-of-m type of option selection. Let $m_{\min}$ and $m_{\max}$ be the minimum and maximum number of options that is allowed to choose to vote.

Setup. To create an election, the EA needs to login to the election server and provides the election definition. An election definition consists of question, options, ($m_{\min}$, $m_{\max}$), start/end time, trustee list (including their email addresses), and voter list (including their voter IDs and email addresses). The election server then creates a unique election ID, eid, selects a bilinear group parameter for the election, $\sigma_{\mathsf{bp}} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{Gen}_{\mathsf{bp}}(1^{\lambda})$, and posts σ_{bp} on BB. (Note that σ_{bp} is hard-coded in our prototype.) The election server then generates and sends a random 128-bit credential to each trustee by email, inviting them to setup the election parameters. Upon receiving the credential, each trustee authenticates himself to the server and executes the election parameters setup process (cf. Section 5.2). Once all the trustees jointly setup the election parameters, the EA triggers the server to send an invitation email (with the voter ID, $\mathsf{vid}_\ell \in \mathbb{G}_2$ and a freshly generated random 128-bit credential s_ℓ) to each voter V_ℓ, where vid_ℓ is a random group element in $\mathbb{G}_2$ generated by the election server.

Cast. After the election starts, each voter V_ℓ uses (vid_i, s_ℓ) to authenticate herself to the election server. Next, she prepares and casts her vote using the voter supporting device VSD. The voters' ballots are prepared locally in the VSD and are posted to the election server, which will be displayed on the BB. (cf. Section 5.3).

Tally. When the election is finished, the server computes the tally ciphertexts by multiplying all the valid submitted ciphertexts for each option on the BB[4]. The voters' coins are used to produce the Sigma protocol challenge (cf. Section 5.4.1). The trustees are then invited by the election server to complete their Sigma protocols and decrypt

[4]Note that, during this step, any invalid ciphertexts and duplicated ciphertexts are removed.

the tally ciphertexts (cf. Section 5.4.2). Note that each trustee should respond to this invitation (EA) using a secure channel such as HTTPS. Upon receiving such a message from a trustee, the election server checks the validity of all the Σ proofs and NIZK proofs, and rejects it in case some of the proofs are invalid. The election server posts all the received trustees' messages to the BB only after all the trustees have successfully completed their **Tally** protocols.

Result. The election result can be computed according to standard threshold ElGamal decryption using the partial decryption of the tally ciphertexts from each trustees.

Verify. After the **Setup** protocol, each trustee T_i is able to check the consistency between the posted election parameters on the BB and its private state st_i. After the election, each voter V_ℓ is able to fetch the election transcript, info from BB and verify the integrity of the election with its receipt rec_ℓ. The voter checks if the data in $\mathbf{blt}_\ell$ hashes to the rec_ℓ and the validity of all the Sigma proofs and NIZK proofs.

5.2 Setup

Generating election parameters. The election parameters generation does not require the interaction between the trustees, and each trustee T_i only needs to interact with the election server. At first, the election server then generates and sends a random 128-bit credential to each trustee inviting them to setup the election parameters. Next, the interaction is completed two rounds. In particular,

Round 1.

- Each trustee T_i performs the following:
  - Pick random $\alpha_i, \beta_i \leftarrow \mathbb{Z}_q$;
  - Set $h_{1,i} = g_1^{\alpha_i}$ and $u_{0,i} = g_1^{\beta_i}$;
  - Post/Append $h_{1,i}, u_{0,i}$ to the public election parameters on the BB together with the following Σ commitment messages:
    - $\Sigma_{(1)}^{\mathsf{dlog}} \{(\alpha_i) : h_{1,i} = g_1^{\alpha_i}\}$;
    - $\Sigma_{(1)}^{\mathsf{dlog}} \left\{(\beta_i) : u_{0,i} = g_1^{\beta_i}\right\}$;
- The election server computes and posts on the BB:
  - $\mathsf{pk} := (g_1, h_1 := \prod_{i=1}^{k} h_{1,i})$;
  - $u_0 := \prod_{i=1}^{k} u_{0,i}$;

Round 2.

- Each trustee T_i performs the following:
  - Pick random $\gamma_i \leftarrow \mathbb{Z}_q$;
  - Set $u_{1,i} = g_1^{\gamma_i}$ and $u_{2,i} = u_0^{\gamma_i}$;
  - Post/Append $u_{1,i}, u_{2,i}$ to the public election parameters on the BB together with the following Σ commitment messages:
    - $\Sigma_{(1)}^{\mathsf{ddh}} \{(\gamma_i) : u_{1,i} = g_1^{\gamma_i} \wedge u_{2,i} = u_0^{\gamma_i}\}$

Upon termination, each trustee T_i keep its working tape as its private state st_i. After all the trustees have participated in **Setup**, the election server:

- Computes $\mathsf{ck} := (u_0, (\prod_{i=1}^{k} u_{1,i}, g_1 \cdot \prod_{i=1}^{k} u_{2,i}))$;
- Posts $\mathsf{crs_m} := (\sigma_{\mathsf{bp}}, \mathsf{ck})$ on the BB;

Generating Voters' Private Information. For every voter V_ℓ, $i \in [n]$, the election server generates (i) the voter ID, $\mathsf{vid}_\ell \in \mathbb{G}_2$ by selecting a random group element in $\mathbb{G}_2$ and (ii) a freshly generated random 128-bit credential s_ℓ. It provides $(\mathsf{vid}_\ell, s_\ell)$, $\ell \in [n]$ to all the voters.

5.3 Cast

The VSD fetches election parameters from the BB and works as a "voting booth". The VSD shows the election question and a list of options to the voter V_ℓ. The voter V_ℓ can select $x \in [m_{\min}, m_{\max}]$ options and let the VSD prepare the ballots. Let $\vec{e} = (e_1, e_2 \ldots, e_m)$ be the characteristic vector corresponding to the voter's selection, where $e_j = 1$ if the option opt_j is selected and $e_j = 0$ otherwise. The VSD prepares two versions of the ballot that encrypts the same option selection as follows

- For $j \in \{1, 2, \ldots, m\}$:
  - Pick random $r_{j,(a)}, r_{j,(b)} \leftarrow \mathbb{Z}_p$;
  - Compute $c_j^{(a)} = (c_{j,1}^{(a)}, c_{j,2}^{(a)}) = (g_1^{r_{j,(a)}}, g_1^{e_j} h_1^{r_{j,(a)}})$;
  - Compute $c_j^{(b)} = (c_{j,1}^{(b)}, c_{j,2}^{(b)}) = (g_1^{r_{j,(b)}}, g_1^{e_j} h_1^{r_{j,(b)}})$;
- Compute two receipts:
  - $\mathsf{rec}_{(a)} = \mathsf{hash}(\mathsf{eid}, \mathsf{vid}_\ell, \text{'A'}, c_1^{(a)}, \ldots, c_m^{(a)})$;
  - $\mathsf{rec}_{(b)} = \mathsf{hash}(\mathsf{eid}, \mathsf{vid}_\ell, \text{'B'}, c_1^{(b)}, \ldots, c_m^{(b)})$;

The VSD presents to the voter the receipts for both A and B versions of the ballot, $\mathsf{rec}_{(a)}$ and $\mathsf{rec}_{(b)}$; meanwhile, it displays two buttons labeled as 'A' and 'B' respectively. The voter should keep the receipts and then randomly choose one of the buttons to proceed. Suppose the voter chooses 'A' (resp. 'B'), the system opens the version B (resp. A) of the ballot by revealing the randomness used to create all the ciphertexts in version B (resp. A), $r_{1,(b)}, \ldots, r_{m,(b)}$. The voter can export the data and use any third-party auditing software to perform the check. The VSD then computes the NIZK proofs for the version A of the ballot; for $j \in \{1, 2, \ldots, m\}$, the VSD computes $\pi_j^{(a)} \leftarrow \mathsf{NIZK}\{\mathsf{crs_m}; (e_j, r_{j,(a)}) : c_j^{(a)} = \mathsf{Enc_{pk}}(e_j; r_{j,(a)}) \wedge e_j \in \{0, 1\}\}$. Note that in above NIZK proofs, $\mathsf{Prov}^{0/1}$ uses the vid_ℓ as the h_2 in the description of Section 4.2.1 instead of generating a fresh $h_2 = g_2^{\alpha_1}$ for $\mathsf{crs}^{(0)}$ and $\mathsf{crs}^{(1)}$ every time. It then sets $c^{(a)} = \prod_{j=1}^{m} c_j^{(a)}$, $e = \sum_{j=1}^{m} e_j$, and $r_{(a)} = \sum_{j=1}^{m} r_{j,(a)}$, and computes $\pi^{(a)} \leftarrow \mathsf{NIZK}\{\mathsf{crs_m}; (e, r_{(a)}) : c^{(a)} = \mathsf{Enc_{pk}}(e; r_{(a)}) \wedge e \in [m_{\min}, m_{\max}]\}$. The VSD posts the ballot $\mathbf{blt}_\ell := \left\langle \mathsf{vid}_\ell, \text{'A'}, \left\{c_j^{(a)}, \pi_j^{(a)}\right\}_{j \in [m]}, \pi^{(a)} \right\rangle$ to the BB.

The voter's receipt is defined as $\mathsf{rec}_\ell := (\mathsf{vid}_\ell, \text{'A'}, \mathsf{rec}_{(a)})$ assuming version A of the ballot was selected during the **Cast** protocol.

5.4 Tally and Result

5.4.1 Producing the Sigma Protocols Challenge

After the election is finished, the voters' coins are collected to produce the Sigma protocol challenge. On the BB, everyone can identify the version of each submitted ballot. We interpret 'A' as 0, 'B' and 1, and if the voter did not submit a ballot, his coin is fixed as 0. Denote ρ_j as the voter V_j's coin and $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$. As studied in [KZZ15], the voters' coins can be modeled as an *adaptive non-oblivious bit fixing source*. Nevertheless, we still want to produce a single challenge if only computationally bounded adversaries are considered. Assume hash is a cryptographic collision resistant hash function such that there is no known algorithm can find a collision with $2^{2\kappa}$ expected steps. We compute the challenge $\mathsf{ch} \leftarrow \mathsf{hash}(\rho)$. In the following theorem, we show that if $H_\infty(\rho) \geq \kappa$, then $H_\infty(\mathsf{hash}(\rho)) \geq \kappa$.

Theorem 4. *Let X be an n-bit efficiently sampleable distribution with $H_\infty(X) \geq \kappa$. Let $\mathsf{hash} : \{0,1\}^* \mapsto \{0,1\}^\lambda$ be a cryptographic hash function, where $\lambda > 2\kappa$ is a security parameter. If $H_\infty(\mathsf{hash}(X)) < \kappa$, then there exists an algorithm that can find a collision for hash within $2^{2\kappa}$ expected number of steps.*

Proof. Since $H_\infty(\mathsf{hash}(X)) < \kappa \leq H_\infty(X)$, we have that

$$\exists \sigma : \Pr[x \leftarrow X : \mathsf{hash}(x) = \sigma] > 2^{-\kappa} \quad \text{and} \quad \forall x : \Pr[x \leftarrow X] \leq 2^{-\kappa}.$$

Therefore, σ must have collisions. Consider the algorithm $\mathcal{A}_\sigma$ that repetitively samples x from X at random and stores $\mathsf{hash}(x)$, trying to find a collision for σ. Given that $\Pr[x \leftarrow X : \mathsf{hash}(X) = \sigma] > 2^{-\kappa}$, the expected running time for $\mathcal{A}_\sigma$ to find a collision for the hash image σ is less than $2^{2\kappa}$. ■

5.4.2 Finalizing the Election

The election server computes the tally ciphertexts by multiplying all the valid submitted ciphertexts for each option on the BB. The tally ciphertexts are denoted by $(E_1, \ldots, E_m)$, where $E_j = (E_{j,1}, E_{j,2})$. Next, each trustee T_i fetches all the posted information from BB and checks its consistency. After that, T_i:

- Computes and sends the following response messages to the election server (EA):
  - $\Sigma^{\mathsf{dlog}}_{(2)} \{(\alpha_i) : h_{1,i} = g_1^{\alpha_i}\}$;
  - $\Sigma^{\mathsf{dlog}}_{(2)} \left\{(\beta_i) : u_{0,i} = g_1^{\beta_i}\right\}$;
  - $\Sigma^{\mathsf{ddh}}_{(2)} \{(\gamma_i) : u_{1,i} = g_1^{\gamma_i} \wedge u_{2,i} = u_0^{\gamma_i}\}$;
- For $j \in \{1, \ldots, m\}$:
  - Computes and sends to the the election server (EA) the partial decryption $D_{j,i} = E_{j,1}^{\gamma_i}$ together with the proof

$$\pi_{i,j} \leftarrow \mathsf{NIZK}\left\{\mathsf{crs_m}; (\gamma_i) : h_{1,i} = g_1^{\gamma_i} \wedge D_{j,i} = E_{j,1}^{\gamma_i}\right\}.$$

After all the trustees partial decryption of the tally ciphertexts has been posted, the election server, for $j \in [m]$, computes $\tau_j = \mathsf{Dlog}_{g_1}(E_{j,2} / \prod_{i=1}^{k} D_{j,i})$. The discrete logarithm can be solved in approximately $\sqrt{n}$ steps, given the knowledge that $\tau_j \in [0, n]$, as there are maximum n possible votes for each option in total. It then posts the final tally $\tau = (\tau_1, \ldots, \tau_m)$ on the BB and informs all the voters by email.

5.5 Verify

After the **Setup** protocol, each trustee T_i is able to check the consistency between the posted election parameters on the BB and its private state st_i. The voter checks the following:

- There is a unique ballot $\mathbf{blt}_\ell$ indexed by vid_ℓ in the election transcript info.
- The data in $\mathbf{blt}_\ell$ hashes to the rec_ℓ.
- There is no duplicated ciphertexts and NIZK proofs across the entire election transcript info.
- All the NIZK proofs in each ballot $\mathbf{blt}_\ell$ uses vid_ℓ as a part of the second layer CRS's.
- All the Sigma and NIZK proofs are valid.

6 Security

6.1 On the Non-malleability of the NIZK Proofs

It is well-known that GS proofs are malleable with respect to the same CRS. More specifically, given a GS proof, π, for the statement x with respect to crs, anyone can re-randomize the proof to produce a distinct proof π^* for x respect to crs. To prevent *replay* attacks, all the duplicated ciphertexts shall be removed. However, the adversary can still copy and re-randomize some honest voters' ciphertexts as well as their attached NIZK proofs if the same CRS is used among all the voters. To address this issue, each voter is required to use a distinct vid_ℓ as a part of her second layer CRS's.

Regarding privacy, recall that we assume the election servers (EA and BB) are honest; in particular, all the voter ID's $\{\mathsf{vid}_\ell\}_{\ell\in[n]}$ should be generated honestly such that no one knows the discrete logarithms: $\mathsf{Dlog}_{g_2}(\mathsf{vid}_\ell)$ for all $\ell \in [n]$ and $\mathsf{Dlog}_{\mathsf{vid}_{\ell_1}}(\mathsf{vid}_{\ell_2})$ for all $\ell_1 \neq \ell_2 \in [n]$. We now show that, given $c = \mathsf{Enc}_{\mathsf{pk}}(b)$ for an unknown $b \in \{0,1\}$ together with a proof π generated by the NIZK proof system $\Gamma^{0/1}$ using vid_1, no PPT adversary can produce $\hat{c} = \mathsf{Enc}_{\mathsf{pk}'}(b)$, where $\mathsf{pk}' \neq \mathsf{pk}$, and $\hat{\pi}$ that includes vid_2 as a part of its second layer CRS's with non-negligible probability.

Recall that in the NIZK proof system $\Gamma^{0/1}$ described in Section 4.2.1 the prover generates $\mathsf{crs}^{(0)}$ and $\mathsf{crs}^{(1)}$ and shows, via a DDH tuple NIZK proof described in 4.1, that the ciphertext c encrypts 0 using $\mathsf{crs}^{(0)}$ and c encrypts 1 using $\mathsf{crs}^{(1)}$. Since the DDH tuple NIZK proof is perfectly sound, if c encrypts b, the proof that uses $\mathsf{crs}^{(b)}$ must be perfectly sound and the proof that uses $\mathsf{crs}^{(1-b)}$ must be simulatable. By the description of the NIZK proof systemΓ^{ddh}, $\mathsf{crs}^{(b)}$ and $\mathsf{crs}^{(1-b)}$ must be encryptions of 1 and 0 respectively under the "public key", $\mathsf{pk}_1 = \mathsf{vid}_1$. Similarly, in $\hat{\pi}$, $\hat{\mathsf{crs}}^{(b)}$ and $\hat{\mathsf{crs}}^{(1-b)}$ must be encryptions of 1 and 0 respectively under the "public key", $\mathsf{pk}_2 = \mathsf{vid}_2$. Hence, the non-malleability problem is reduced to the following theorem.

Theorem 5. *Given randomly chosen* $\mathsf{pk}_1, \mathsf{pk}_2$ *and* $c_0 = \mathsf{Enc}_{\mathsf{pk}_1}(m)$, $c_1 = \mathsf{Enc}_{\mathsf{pk}_1}(1-m)$ *for unknown* $m \in \{0,1\}$, *the probability that a PPT adversary* $\mathcal{A}$ *produces* $\hat{c}_0 = \mathsf{Enc}_{\mathsf{pk}_2}(m)$, $\hat{c}_1 = \mathsf{Enc}_{\mathsf{pk}_2}(1-m)$ *is negligible ,if the underlying encryption scheme is IND-CPA secure.*

Proof. The proof is via reduction. Assume there is a PPT adversary $\mathcal{A}$ who can produce $\hat{c}_0 = \mathsf{Enc}_{\mathsf{pk}_2}(m)$, $\hat{c}_1 = \mathsf{Enc}_{\mathsf{pk}_2}(1-m)$. Then, we can contruct an adversary $\mathcal{B}$ who can win the IND-CPA game of the underlying encryption scheme as follows:

In the IND-CPA game, $\mathcal{B}$ is given pk_1 and it sends $m_0 = 0, m_1 = 1$ to the IND-CPA challenger $\mathcal{C}$. $\mathcal{B}$ will receive $c_0 = \mathsf{Enc}_{\mathsf{pk}_1}(m_b)$ from $\mathcal{C}$ and will be challenged to guess b. $\mathcal{B}$ then computes $c_1 = \mathsf{Enc}_{\mathsf{pk}_1}(1)/c_0 = \mathsf{Enc}_{\mathsf{pk}_1}(1-m_b)$ and generates $(\mathsf{sk}_2, \mathsf{pk}_2)$. Next, it sends c_0, c_1, pk_1 and pk_2 to $\mathcal{A}$. Upon receiving $\hat{c}_0$ and $\hat{c}_1$ from $\mathcal{A}$, $\mathcal{B}$ decrypts $\hat{c}_0$ and $\hat{c}_1$. Finally, it sends b' to $\mathcal{C}$, if $\hat{c}_0$ and $\hat{c}_1$ are indeed encryptions of b' and $1-b'$, otherwise, she sends random $b' \leftarrow \{0,1\}$ to $\mathcal{C}$.

Clearly, $\mathcal{B}$ wins when $\mathcal{A}$ succeeds (i.e., $\hat{c}_0$ and $\hat{c}_1$ are indeed encryptions of b' and $1-b'$). Therefore if the probability that $\mathcal{A}$ wins is p, we have that

$$\begin{aligned}\Pr[\mathcal{B} \text{ wins}] &= \Pr[\mathcal{A} \text{ succeeds}] \cdot \Pr[\mathcal{B} \text{ wins} | \mathcal{A} \text{ succeeds}] + \Pr[\mathcal{A} \text{ fails}] \cdot \Pr[\mathcal{B} \text{ wins} | \mathcal{A} \text{ fails}] = \\ &= p \cdot 1 + (1-p) \cdot 1/2 = 1/2 + p/2.\end{aligned}$$

■

6.2 Privacy

Our system achieves the simulation-based privacy defined in Section 3.2. Similarly to [KZZ15], we utilize complexity leveraging. Specifically, we choose the security parameters such that breaking the SXDH assumption of $\mathsf{Gen}_{\mathsf{bp}}$ and finding a collision for hash is much harder than guessing the challenge of the Sigma protocols.

Theorem 6. *Assume there exists a constant κ, $0 < \kappa < 1$ such that for any 2^{λ^κ}-time adversary $\mathcal{A}$ the advantage of breaking the SXDH assumption of $\mathsf{Gen}_{\mathsf{bp}}$ is $\mathsf{negl}(\lambda)$. Let $n, m, t, k \in \mathbb{N}$, where $0 < k < n$. Then, for every $m, n, t = poly(\lambda)$ and every $k < \lambda^\kappa$, the e-voting system Π described in Section 5 is k-private, unless there is an explicit algorithm that can find a collision for $\mathsf{hash} : \{0,1\}^* \mapsto \{0,1\}^\lambda$ in 2^{λ^κ} time.*

Proof. (Sketch) Given an adversary $\mathcal{A}$, we construct a simulator $\mathcal{S}$ s.t. $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{priv}}^{m,n,t}, \mathcal{S}, \mathcal{Z}}(\lambda)$ and $\mathsf{REAL}_{\Pi, \mathcal{A}, \mathcal{Z}}(\lambda)$ are computationally ingistinguishable. Without loss of generality, let T_w be the honest trustee. The simulator $\mathcal{S}$ operates as follows:

At the beginning of the experiment, $\mathcal{S}$ selects all the voters' coins (including both honest and corrupted voters) at random, denoted as $\rho = (\rho_1, \ldots, \rho_n) \in \{0,1\}^n$ and produces the challenge of the Sigma protocols using ρ. When $\mathcal{S}$ receives $(sid, \mathsf{vote}, \mathcal{O}, \mathcal{V}, \mathcal{U})$ from $\mathcal{F}_{\mathsf{priv}}^{m,n,t}$, it simulates Π in the **Setup** protocol playing roles as EA and BB, and interacting with all the corrupted trustees. In addition, it generates $\mathcal{T} = \{T_1, \ldots, T_w\}$ and allows $\mathcal{A}$ to corrupt all the trusteed except from T_w. In the simulation of T_w, $\mathcal{S}$ performs the following modifications: it sets $u_{2,(w)} = u_0^{\gamma_w}/g_1$ and simulates a proof for the fake DDH relation of $(g_1, u_0, u_{1,(w)}, u_{2,(w)})$. Once all the trustees have completed their **Setup**, $\mathcal{S}$ generates vid_ℓ such that $d_\ell = \mathsf{Dlog}_{g_2}(\mathsf{vid}_\ell)$ is known. It then sends the credentials to all the voters.

During the **Cast** protocol, $\mathcal{S}$ plays the role of the EA and BB. Upon receiving $(sid, \mathsf{cast}, V_\ell)$ from $\mathcal{F}_{\mathsf{priv}}^{m,n,t}$ for an honest V_ℓ, $\mathcal{S}$ executes a **Cast** protocol on behalf of V_ℓ with a random $\mathcal{U}_\ell \in \mathcal{U}$. After all the voters cast their ballots, $\mathcal{S}$ plays the role of the EA interacting with the corrupted trustees in the **Tally** protocol. Importantly, $\mathcal{S}$ sends suitably long messages to EA to fake the **Tally** interaction for T_w. Due to the secure channel between T_w and the EA, $\mathcal{A}$ cannot tell T_w's **Tally** protocol is fake.

After all the corrupted trustees finish the **Tally** protocol, $\mathcal{S}$ does not post their tally messages to the BB; instead, it stores the set of the transcripts of all the Sigma protocols and rewinds the state of the experiment to the **Cast** protocol of the last honest voter, V_L. In the second run, $\mathcal{S}$ executes the **Cast** protocol for V_L again but this time it chooses a different ballot version to submit. Namely, $\mathcal{S}$ flips the coin of V_L. Then, $\mathcal{S}$ completes the rest of the protocol in the second run until all the corrupted trustees finish the **Tally** protocol. If there is no collision for hash, the challenge of the Σ protocols must be distinct from that of the first run, otherwise we obtain a collision finding algorithm for hash. Hence, $\mathcal{S}$ obtains another set of the transcripts of all the Σ protocols with a different challenge. Subsequently, $\mathcal{S}$ utilizes the knowledge extractor to extract all the corrupted trustees' witnesses $\alpha_i, \beta_i, \gamma_i$, $i \neq [k] \setminus \{w\}$. Hence, $\mathcal{S}$ can compute $\gamma = \sum_{i=1}^{k} \gamma_i$. Note that now the master CRS $\mathsf{crs_m}$ is an encryption of 0 and thus it is a perfectly simulatable CRS and γ is the trapdoor.

After that, $\mathcal{S}$ rewinds the state of the experiment to the beginning of the **Cast** protocol and starts a third run. In the **Cast** protocol, $\mathcal{S}$ uses the pre-generated coins ρ_ℓ of each honest voter V_ℓ. In addition, it encrypts an invalid $\mathcal{U}_\ell^* \notin \mathcal{U}$ and simulates the NIZK proofs using the trapdoor γ. In case the corrupted voters' coins do not match the pre-generated (guessed) coins, $\mathcal{S}$ resets back to the beginning of the experiment and starts over. $\mathcal{S}$ repeats the above procedure until it has three runs of the execution and the voters' coins of the third run execution is guessed correctly. The expected running time to make this happen is $2^k \cdot poly(\lambda) < 2^{\lambda^\kappa}$.

Subsequently, for each corrupted voter V_ℓ, $\mathcal{S}$ uses d_ℓ to decrypt all the second layer CRSs in her ballot $\mathbf{blt}_\ell$ on the BB, and thus determine $\mathcal{U}_\ell$. If $\mathcal{U}_\ell \notin \mathcal{U}$, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ sends $(sid, \mathsf{cast}, V_\ell, \mathcal{U}_\ell)$ to $\mathcal{F}_{\mathsf{priv}}^{m,n,t}$. After sending all the corrupted voters' option selections, $\mathcal{S}$ sends (sid, tally) to $\mathcal{F}_{\mathsf{priv}}^{m,n,t}$. Upon receiving the election result $\tau := (\tau_1, \ldots, \tau_m)$ from $\mathcal{F}_{\mathsf{priv}}^{m,n,t}$, for $j \in [m]$, $\mathcal{S}$ computes $D^*_{j,(w)} = E_{j,2}/(g_1^{\tau_j} \cdot E_{j,1}^{\sum_{i \neq w} \alpha_i})$ and simulates the NIZK corresponding proofs. Finally, $\mathcal{S}$ posts $D^*_{j,(w)}$ on the BB.

We first show that the probability that the above simulator $\mathcal{S}$ aborts is negligible. In case the extracted $\mathcal{U}_\ell \notin \mathcal{U}$ for some corrupted voter V_ℓ, the adversary $\mathcal{A}$ must have managed to either break the soundness of the underlying NIZK proof system or 'copy' one of the honest voter's ciphertexts by re-randomizing them. According to Theorems 2, 3 and 5, either events happen with negligible probability.

Suppose now $\mathcal{S}$ does not abort. We will show that if the lifted ElGamal is IND-CPA secure, then the protocol view created by $\mathcal{S}$ is indistinguishable from the real execution. Note that IND-CPA security of the ElGamal implies that SXDH assumption holds. Given an adversary $\mathcal{A}$ who can distinguish the protocol view simulated by $\mathcal{S}$, we can construct an adversary $\mathcal{B}$ who can break the IND-CPA game. Indeed, in the reduction, when $\mathcal{B}$ receives the public key, we will post it as $h_{i,(w)}$ in the **Setup** protocol, simulating the Dlog Sigma protocol. $\mathcal{B}$ then sends $m_0 = 0, m_1 = 1$ to the IND-CPA challenger. When receiving a ciphertext $c = (c_1, c_2)$, $\mathcal{B}$ can transfer the ciphertext under public key $h_{1,(w)}$ to be a ciphertext under the public key h_1 and use it in the honest voters' ballots. The transformation: $c' = (c_1, c_2 \cdot g_1^{\sum_{i \neq w} \alpha_i})$. Clearly, c and c' encrypts the same message under different public keys. If the adversary $\mathcal{A}$ can distinguish the honest voters' ballots, then the adversary $\mathcal{B}$ distinguish the IND-CPA challenge with running time $2^k \cdot poly(\lambda) < 2^{\lambda^c}$. ■

Remark. As in [KZZ15], we use complexity leveraging to argue privacy which means

$k < \lambda$. But for any desired k we can always choose a suitable security parameter λ such that the system is k-private. In most real world elections (e.g., national elections) privacy is only guaranteed between hundreds or a few thousands voters that belong to a precinct. If one wants to achieve privacy nation-wide as well, it is still possible to use our scheme efficiently with the following modification: the trustees, each one individually, will perform a Sigma OR proof that either their published parameter is properly generated or that they know a preimage of a one-way hash function of the coins of the voters (this should be done using a Lapidot-Shamir like proof since the statement is not determined fully before the first move of the protocol). In the privacy proof the simulator can use complexity leveraging to find such preimage in time independent of the number of corrupted voters and thus complete the simulation in time proportional to breaking the one-way function.

6.3 E2E Verifiability

For simplicity, our analysis is for 1-out-of-m elections can be easily extended to x-out-of-m cases. Our proof strategy follows the lines in [KZZ15, Theorem 4], as our e-voting system shares many common elements with DEMOS.

Theorem 7. *Let $n, m, t, \theta d \in \mathbb{N}$ where $1 \leq \theta \leq n$. The e-voting system described in Section 5 achieves E2E verifiability for a number of θ honest successfull voters and tally deviation d with error $(1/2)^d + (1/2)^\theta$ unless there is an algorithm that can find a collision for $\mathsf{hash} : \{0,1\}^* \mapsto \{0,1\}^\lambda$ in $2^{2\theta}$ expected number of steps.*

Proof. We emphasize that in the E2E verifiability proof, only BB is assumed to be honest. The rest components of the election server are controlled by the adversary. Hence, the voter ID, vid_ℓ, may not necessarily be unique, and the adversary is allowed to change the content on the BB arbitrarily before the **Tally** protocol starts. Nevertheless, we can assume all the Sigma protocols and the NIZK proofs on the BB are valid if there is at least one honest voter that performs verification. We first construct a vote extractor $\mathcal{E}$ for our system as follows:

$\mathcal{E}$ takes input as the election transcript, info and a set of receipts $\{\mathsf{rec}_\ell\}_{V_\ell \in \tilde{\mathcal{V}}}$. If $\mathbf{Result}(\mathsf{info}) = \perp$, then $\mathcal{E}$ outputs $\perp$. Otherwise, for all the corrupted voters $V_\ell \in \mathcal{V} \setminus \tilde{\mathcal{V}}$, $\mathcal{E}$ extracts $\mathcal{U}_\ell$ by exhaustive search over the ElGamal ciphertexts in the ballot $\mathbf{blt}_\ell$. It then outputs $\langle \mathcal{U}_\ell \rangle_{V_\ell \in \mathcal{V} \setminus \tilde{\mathcal{V}}}$.

Based on the above vote extractor, we now prove the E2E verifiability of our scheme. Assume an adversary $\mathcal{A}$ that wins the game $G^{\mathcal{A},\mathcal{E},d,\theta}_{\mathsf{e2e\text{-}ver}}(1^\lambda, n, m, t)$. Namely, $\mathcal{A}$ breaks E2E verifiability by allowing at least θ honest successful voters and achieving tally deviation d. Let E be the event that there exists one tallied ciphertext that encrypts $e^* \notin \mathcal{U}$. By Theorems 1 and 2, all the NIZK proofs are perfectly sound. Hence, the adversary needs to cheat on at least one Sigma protocol to make event E occur. By Theorem 4 and since the voters' coins have min entropy θ, the Sigma protocols challenge $\mathsf{hash}(\rho)$ should also have min entropy θunless there is an algorithm that can find a collision for hash in $2^{2\theta}$ expected number of steps. Hence, each Sigma protocol has soundness error no more than $(1/2)^\theta$. Therefore,

$$\Pr[G^{\mathcal{A},\mathcal{E},d,\theta}_{\mathsf{e2e\text{-}ver}}(1^\lambda, n, m, t) = 1 \mid E] \leq (1/2)^\theta. \tag{1}$$

Now assume that E does not occur. In this case, the deviation from the intended result that $\mathcal{A}$ achieves, derives only by miscounting the honest votes. This may be achieved

by $\mathcal{A}$ in two different possible ways:

- **Modification attacks.** Modify one of the versions of the honest voters' ballots when it was produced on the VSD. This attack is successful only if the voter chooses to submit the modified version. Since the version is valid, it encrypts an option in $\mathcal{U}$, hence for each successful attack the achieved deviation is 1.
- **Clash attacks.** Assign the same vid to y honest voters so that the adversary can inject $y-1$ ballots. This attack is successful only if all the y voters verify the same ballot on the BB and hence miss the injected votes that produce the tally deviation. The maximum deviation achieved by this attack is $y-1$.

Recall that each honest voter should select one of the two versions of the ballot at random, and the other version will be opened for auditing. Hence, the success probability of x deviation via the modification attack is $(1/2)^x$. With regard to the clash attacks, similarly, it is easy to see that the success probability to clash y honest voters without being detected is $(1/2)^{y-1}$ (all y honest voters choose the same version to vote). Given that E does not occur the total tally deviation achieved is $x+y \geq d$. Therefore, the upper bound of the success probability of $\mathcal{A}$ when E does not occur is

$$\Pr[G^{\mathcal{A},\mathcal{E},d,\theta}_{\mathsf{e2e\text{-}ver}}(1^\lambda, n, m, t) = 1 \mid \neg E] \leq (1/2)^{x+y} \leq (1/2)^d. \tag{2}$$

By Eq. (1), (2), we have the overall probability

$$\Pr[G^{\mathcal{A},\mathcal{E},d,\theta}_{\mathsf{e2e\text{-}ver}}(1^\lambda, n, m, t) = 1] \leq (1/2)^d + (1/2)^\theta \ .$$

■

7 Implementation

Our prototype is written in Django framework. We also adopt Twitter Bootstrap [Boo15] for better user interface. All the cryptographic elements are Base64 encoded and interchanged in JSON format. We use SHA3 to implement hash and adopt CryptoJS [Mot15] as its JavaScript implementation. We use Type F pairing groups [BN06] as the asymmetric bilinear group candidates. Its JavaScript implementation employs SJCL [SJC15] for basic big number arithmetic. On top of SJCL, we ported the Type F pairing arithmetic of jPBC [DI11] to JavaScript. Unlike DEMOS [KZZ15], the election setup step of DEMOS-2 is very efficient and independent of m, n. This is because all the votes are encrypted at the client side during the cast phase on demand. The benchmark results in Tbl. 1 shows the time that a VSD (client) takes to encrypt a vote and produce a ballot (using Javascript). The benchmark is produced on a Mac Mini with 2.5 GHz Intel Core i5, 4GB RAM. For instance, when $m = 2$, it takes 0.4s to prepare both A and B versions of the ballot; afterwards, it takes 2.2s to complete the NIZK proofs.

References

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX*, 2008.

m	Security	Version A&B	NIZK proof	Ballot Size
2	80 bits	399.4 ms	2239.2 ms	2.5 KB
10	80 bits	1913.5 ms	8210.4 ms	9.3 KB

Table 1: Client-side Vote Encryption Efficiency

[BCP+11] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting helios for provable ballot privacy. In *ESORICS*, 2011.

[Ben06] Josh Benaloh. Simple verifiable elections. In *USENIX*, 2006.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC*, 1988.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *SAC*, pages 319–331. Springer, 2006.

[Boo15] Bootstrap. Twitter Bootstrap. `http://getbootstrap.com/`, 2015.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *ASIACRYPT*, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73, 1993.

[Can98] Ran Canetti. Security and composition of multi-party cryptographic protocols. *J. of CRYPTOLOGY*, 13:2000, 1998.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

[Cha90] David Chaum. Zero-knowledge undeniable signatures (extended abstract). In *EUROCRYPT*, pages 458–464. Springer, 1990.

[DI11] Angelo De Caro and Vincenzo Iovino. jpbc: Java pairing based cryptography. In *ISCC 2011*, pages 850–855, 2011.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *CRYPTO*, pages 97–111, 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008.

[JCJ02] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. *ePrint*, 2002:165, 2002.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and relationship to verifiability. *ePrint*, 2010:236, 2010.

[KTV11] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, privacy, and coercion-resistance: New insights from a case study. In *S & P*, pages 538–553, 2011.

[KZZ15] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In *EUROCRYPT*, pages 468–498. Springer, 2015.

[LS90] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*. Springer, 1990.

[Mot15] Jeff Mott. Crypto-JS. `http://code.google.com/p/crypto-js/`, 2015.

[Ràf15] Carla Ràfols. Stretching groth-sahai: NIZK proofs of partial satisfiability. In *TCC*, 2015.

[RG15] Mark Ryan and Gurchetan S. Grewal. Online voting is convenient, but if the results aren't verifiable it's not worth the risk. The Conversation, `https://theconversation.com/online-voting-is-convenient-but-if-the-results-arent-verifiable-its-not- worth-the-risk-41277`, May 2015.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *EUROCRYPT*, volume 434, pages 688–689. Springer, 1989.

[SJC15] SJCL. Stanford Javascript Crypto Library. `http://crypto.stanford.edu/sjcl/`, 2015.