FiltrumVote: Scalable, Verifiable, and Coercion-Resistant Internet Voting

Leonardo Kimura[1], Dimitri Leskow[1], Diego F. Aranha[3], Roberto Araújo[2], and Marcos Simplicio[1]

[1] University of Sao Paulo, Brazil lkimura,mjunior}@larc.usp.br
[2] Federal University of Para, Brazil
[3] Aarhus University, Denmark

Abstract. Internet voting can increase voter turnout and reduce operational costs; however, it also facilitates voter coercion. One compelling approach to address this issue is deniable vote updating, which allows voters to cast multiple ballots while counting only their final vote. Despite its intuitive appeal, achieving deniable vote updating together with verifiability and scalability has proven challenging. Therefore, we present FiltrumVote, a scalable, publicly verifiable, and coercion-resistant Internet voting system based on deniable vote updating. It has a quasi-linear cleansing phase, and it is built mainly from zero-knowledge proofs with standard $\Sigma-$protocol techniques. As a result, FiltrumVote is highly scalable: a standard computer takes less than 5 hours to prove all zero-knowledge proofs for one million votes.

Keywords: Coercion-resistant voting · End-to-end verifiability · Encrypted sorting · Range proofs

1 Introduction

Internet voting has the potential to increase voter turnout, broaden accessibility, and reduce costs. Consequently, numerous protocols have been proposed for secure Internet-based elections, using cryptographic techniques to balance privacy, verifiability, and correctness. However, one critical requirement for secure elections is coercion-resistance: adversaries should not be able to pressure voters to vote for particular candidates. The remote nature of Internet voting significantly increases the risk of coercion compared to traditional, in-person voting environments.

A variety of approaches have been developed to mitigate coercion in Internet-based election systems. One of the earliest and most influential strategies is the fake-credential mechanism, first introduced in JCJ [16]. In this model, voters can generate fake credentials that are indistinguishable from their real ones, so that fake credentials can be handed to a coercer, while genuine credentials are used to cast a valid ballot. However, this model presents significant usability challenges, since voters must confidently provide only the fake credentials to an adversary, while remembering to use the legitimate credentials only to cast valid

votes. Although later works improved scalability, usability, and security of this approach [5, 4, 6], empirical studies show that the burden on usability is still substantial [7, 25].

An alternative coercion-resistance strategy that has gained considerable attention is deniable vote updating (or revoting) [25]. In this model, voters can cast their ballots multiple times, and only the final vote is counted. This provides an intuitive strategy to resist coercion, and has inspired several real-world protocols, including the Internet voting system used in Estonia [27]. Yet achieving deniable vote updating while maintaining verifiability is challenging. In particular, the system must not reveal how many times an individual voter has voted; otherwise, a coercer could confirm whether a voter has updated a ballot or not after the coercion. Thus, traditional tallying mechanisms, which publish all valid votes, are inadequate. Existing solutions attempt to reconcile these requirements using noise votes [20, 13], credential-hiding cleansing [1, 19], or periodic vote updating [24]. However, these designs either impose substantial computational overhead [1, 19, 24] or rely on strong trust assumptions [13].

Our contribution. We propose FiltrumVote, an Internet voting protocol that simultaneously achieves verifiability, scalability, and coercion resistance through deniable vote updating. While prior work has explored verifiable deniable vote updating, to the best of our knowledge, FiltrumVote is the first protocol to scale to more than one million votes using only modest computational resources. The prover time, verification time, and proof size all grow linearly with the number of votes, with the sole exception of the sorting phase, which relies on a standard quasi-linear sorting algorithm that can be performed in plaintext. FiltrumVote is inspired by CHide [6, 3], an enhancement of the JCJ voting system that employs a less revealing tally procedure. We adapt this approach to support revoting, thus reducing the responsibilities of trustees and substantially lowering both system complexity and computational overhead.

2 Related Works on Deniable Vote Updating

Revoting, or the ability for voters to update their votes, is already supported in many real-world deployed systems, such as Helios [2] and the Estonian Voting System (IVXV) [14]. Thus, voters have an intuitive mechanism to mitigate coercion attacks, offering clear usability advantages over alternative approaches (e.g., fake credentials) [25]. However, these systems provide only superficial protection against coercion. For example, information such as signed tracking codes or transaction logs may reveal how many votes a voter has cast [27]. Consequently, voters cannot hide whether they have updated their vote — a property known as *deniable vote updating*. As a result, a determined coercer can verify whether an attempted coercion was successful.

The first works that provide both verifiability and deniable vote updating are [1] and [19], which use plaintext equivalence tests (PETs) to enable a credential-hiding tallying procedure. In these schemes, distributed authorities compare all

pairs of ballots and privately mark only the last ballot associated with each credential. However, this approach results in an $O(n^2)$ protocol, making it suitable only for small elections.

A subsequent work is VoteAgain [20], which hides ballots in a swarm of *dummy* ballots. Dummy and overwritten ballots are secretly yet verifiably canceled, leaving only valid ballots in the final tally. This approach results in a quasi-linear cleansing time $O(n \log(n))$. However, VoteAgain relies on strong trust assumptions to achieve public verifiability. In particular, even in the improved protocol [13], vote integrity holds only if there is no malicious collusion between the polling authority and the tally server. Otherwise, votes can be modified or discarded without detection.

Another approach by Farzaliyev and Willemson is introduced in [11], which combines private bulletin boards with vector commitments and Succinct Non-Interactive Argument of Knowledge (SNARKs). This results in a solution with extremely reduced verification time and a relatively small proof size. However, this efficiency comes at the cost of generating SNARK proofs. For example, processing 2^{24} votes would require a peak memory usage of approximately 500 GB. In addition, the system requires 16 TB to store some individual proofs, which requires 41 core-days to verify sequentially.

Finally, DeVoS achieves coercion-resistance by employing periodic vote updating [24]. In each microsubmission interval, all votes are reencrypted or updated in zero-knowledge, thus hiding whether the voter has revoted or not. This results in a cleansing time linear to the number of votes $O(n)$. However, DeVoS' scalability is limited by the need to generate many zero-knowledge proofs in *every* microsubmission interval. For example, a server with 16 GB of RAM can generate proofs for only $\approx 10,000$ to $80,000$ votes within a two-minute microsubmission interval. In addition, this estimate assumes only four candidates; supporting additional candidates further increases the computational cost. Even with optimizations, handling elections with over one million votes would require substantial computational resources.

3 FiltrumVote Model

In this section, we present a high-level description of FiltrumVote.

3.1 System and Threat Model

The participants of the FiltrumVote protocol are:

- *Election Authority (EA).* The entity responsible for conducting the election, including registering and authenticating voters, receiving, and processing ballots.
- *Trustees.* A set of authorities that mix and decrypt the votes.
- *Public Bulletin Board (PBB).* An append-only data structure where the EA publishes the list of received votes, and where both the EA and the trustees

publish processed votes and associated proofs. The contents of the PBB are publicly readable at all times.
- *Voters.* Individuals who cast their ballots by interacting with the EA.
- *Auditors.* Independent parties responsible for verifying the correctness of the election, including the validity of the zero-knowledge proofs.

FiltrumVote assumes the following key trust assumptions:

1. The PBB is honest, that is, it never removes data and provides a consistent view of the board to all participants. Several approaches can be used to achieve this (e.g., [17, 15, 8]).
2. Voters securely store and protect their private keys.
3. Voters verify that their public keys are correctly registered.
4. There exists a public list of eligible voters (voter roll), and any participant can verify its correctness.
5. The voting device is fully trusted with respect to both integrity (e.g., through a second verification device [21]) and privacy.
6. The voter can cast at least one vote after coercion attempts and before the end of the election.

Although the second assumption is strong, it is standard in the state-of-the-art Internet voting protocols such as DeVos and JCJ. One approach to protect the keys is to store them on secure hardware devices; however, this facilitates forced abstention attacks, as the coercer may simply confiscate the device. Another mitigation is to reduce the risk of voter impersonation by requiring extra authentication with the EA when casting a vote (e.g., biometrics), so that the private key alone is not enough to cast a vote. Fully reducing this trust assumption remains an open problem.

We also assume that a voter can register their public key with the EA through an authenticated channel. In addition, we assume the voter can send one vote to the EA through a private channel, so that the adversary cannot learn whether or when a vote has been cast. These assumptions are standard in Internet voting protocols, such as DeVos.

Finally, we require our system to satisfy the following properties, defined informally:

- *Ballot secrecy.* Assuming a threshold number of trustees are honest, no collusion of malicious parties can learn the vote of any honest voter.
- *Verifiability.* Any party can verify that the reported election outcome is correct, even under collusion between a malicious EA and malicious trustees. In particular, ballot-stuffing attacks are detectable.
- *Coercion-resistance.* Assuming an honest EA and a threshold number of honest trustees, no coercer can determine whether a coercion attack was successful. In particular, forced abstention attacks are not possible.

Notably, we assume that the EA is honest with respect to coercion resistance. A malicious EA could carry out a successful coercion attack, for example,

by learning how many times a voter has cast a ballot or by refusing to accept new votes after a coerced one. To mitigate this risk, the cleansing phase must be conducted privately (maybe in a controlled environment without human intervention), and any refusal to accept new votes should be investigated. This assumption is standard in the literature: other Internet voting protocols, such as DeVos and VoteAgain, likewise have a party fully trusted with coercion resistance.

3.2 High-level Overview

Here, we provide a high-level overview of FiltrumVote. Intuitively, voters submit their ballots along with their encrypted credentials. The cleansing phase uses the credentials to identify and remove overwritten votes in five steps. Each step is done privately and verifiably using zero-knowledge proofs.

We denote by $[\![x]\!]$ an encryption of x under the EA's public key, and by $[\![x]\!]_T$ an encryption of x under the threshold-generated trustees' public key.

Registration phase. Each voter with identifier vid_i generates a key pair (sk_i, pk_i) and sends their public key pk_i to the EA through an authenticated channel. The EA then assigns to the voter a unique credential number σ_i, and publishes the voter roll (vid_i, pk_i, σ_i). Auditors check if every σ_i is unique and every vid_i is from an eligible voter; voters check if their public key pk_i is correct.

Submission phase. The voter V_i encrypts her individual vote v_i under the trustees' public key pk_T, resulting in $[\![v_i]\!]_T$. She also encrypts her public key pk_i and her credential number σ_i under EA's public key, resulting in $[\![pk_i]\!], [\![\sigma_i]\!]$. Then, the voter provides an *non-interactive zero-knowledge proof (NIZKP) of possession* π_{poss} (Section 4.1). This proof essentially ensures that the voter knows the secret key associated with the encrypted public key; i.e., that she knows sk_i. The voter also computes an NIZKP of plaintext knowledge π_{kw} for $[\![v_i]\!]_T$ and $[\![\sigma_i]\!]$. The resulting ballot $\beta = ([\![v_i]\!]_T, [\![pk_i]\!], [\![\sigma_i]\!], \pi_{poss}, \pi_{kw})$ is sent to the EA. The EA checks the ballot (its proof and whether it is unique) and sends it to the public bulletin board, PBB. Voters can vote multiple times, and only the last vote will be counted.

Note that π_{poss} is quite a permissive proof: it only proves that the ciphertext encrypts *a public key*, and the voter knows the corresponding secret key. Thus, a malicious voter may use a new, ephemeral public key or add to the ballot an encrypted credential number totally unrelated to her credential. At this point, we need to consider two kinds of attacks.

First, an *impersonation attack* may use a credential σ_i from a valid voter, while using a public key different from that of the valid voter, i.e., $pk' \neq pk_i$. To protect against it, we require the EA to check every submitted ballot during the submission phase. Thus, the EA must decrypt the $[\![pk']\!]$ and $[\![\sigma']\!]$ of submitted ballots, and only accept them if the pair (pk', σ') is present in the voter roll.

Note that EA is still considered untrusted; the correct behavior of the EA can be confirmed in the *consistency check step* (explained later).

Second, a *ballot stuffing attack* may use a credential σ' not existent in the voter roll. To protect against it, the *selection step* ensures that only ballots from eligible voters are considered (i.e., with credentials present in the voter roll). Thus, all other votes are verifiably canceled, even by a malicious EA. However, an honest EA might also reject these invalid votes in the submission phase to reduce the number of votes to be processed later.

Cleansing phase - preparation. Once the voting phase is finished, the EA starts the cleansing phase by processing all entries. First, the EA takes the submitted ballots $\beta_i = (\llbracket v_i \rrbracket_T, \llbracket pk_i \rrbracket, \llbracket \sigma_i \rrbracket)$ from the PBB in chronological order. Each ballot is appended with an encrypted zero (acting as a flag, $\llbracket f_i \rrbracket = \llbracket 0 \rrbracket$) and an encrypted sequential counter ($\llbracket ctr_i \rrbracket$).

Next, the EA augments this list by generating "flag tuples" derived from the voter roll. For each entry in the voter roll, the EA encrypts the voter's public key pk_i and credential σ_i. Then, it appends it with an encrypted dummy vote ($\llbracket v_i \rrbracket_T = \llbracket 1 \rrbracket$), an encrypted flag set to one ($\llbracket f_i \rrbracket = \llbracket 1 \rrbracket$), and the continuing encrypted counter ($\llbracket ctr_i \rrbracket$).

At the end, the actual ballots and flag tuples are in the format $t_i = (\llbracket v_i \rrbracket_T, \llbracket pk_i \rrbracket, \llbracket \sigma_i \rrbracket, \llbracket f_i \rrbracket, \llbracket ctr_i \rrbracket)$. Figure 1 illustrates the resulting board.

Cleansing phase - mixing. The EA mixes the list of tuples using a verifiable shuffle protocol.

Cleansing phase - sorting. The EA decrypts the credential σ_i, marker f_i, and the counter ctr_i of each tuple, and performs a sorting algorithm on the mixed list of tuples. To do so, we use the following relationship: for two tuples, t_i and t_j, we consider $t_i < t_j$ when the binary string $\sigma_i || f_i || ctr_i$ is less than $\sigma_j || f_j || ctr_j$.

This results in a list in which tuples from the same credentials are grouped and sorted chronologically. In addition, if a ballot was cast with an authorized credential (i.e., σ_i is in the voter roll), the last tuple from each credential group is a "flag tuple" (with $f_i = 1$).

Note that the EA can decrypt σ_i, f_i, ctr_i to perform the sorting in plaintext; however, it cannot reveal the plaintexts without compromising coercion resistance. Thus, after the sorting, the EA will provide a *NIZKP of greater-than* for every two consecutive tuples (Section 4.1). External auditors can check that the encrypted set is correctly sorted by verifying these proofs. Note that it is safe to leak the comparison result $t_i < t_j$ due to the previous mixing phase.

Cleansing phase - consistency check. The EA proves that there are no impersonation attacks in the list by providing a *NIZKP of consistency* between every two consecutive tuples (Section 4.1). This proof shows that either both

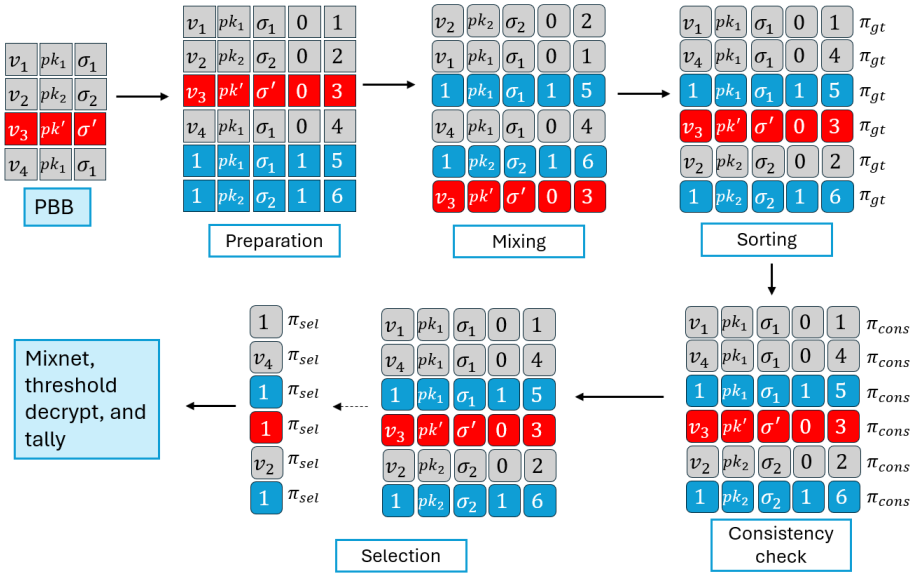

Fig. 1: High-level overview of the FiltrumVote cleansing phase. In this example, three tuples corresponding to valid votes are present: two submitted by voter (pk_1, σ_1) and one by voter (pk_2, σ_2). There is also a ballot-stuffing attempt (pk', σ'), shown in red. The blue entries correspond to the "flag tuples" generated from the voter roll.

public keys and credentials are equal, or both are different, i.e., $(pk_i = pk_j \wedge \sigma_i = \sigma_j) \vee (pk_i \neq pk_j \wedge \sigma_i \neq \sigma_j)$.

To see that these *NIZKPs* effectively prove that no impersonation attacks are present in the list, consider the following: a tuple t' representing an impersonation attack has (1) an existing credential σ_i, and (2) a public key $pk' \neq pk_i$. Because of the sorting phase, all credentials are grouped: so t' is consecutive to some tuple t with credential σ_i and public key pk_i. Since $\sigma_i = \sigma_i$ and $pk' \neq pk_i$, it is impossible to provide a correct *NIZKP of consistency* for this tuple.

Cleansing phase - selection. For each tuple in the list, the EA generates a new encrypted vote V_i' that is either (1) a reencryption of $[\![v_i]\!]_T$, if the tuple is valid; or (2) an encryption of 1, if the tuple is invalid. Valid tuples are defined as those that immediately precede their corresponding "flag tuple" (i.e., $\sigma_1 = \sigma_2 \wedge f_2 = 1$). To demonstrate that each V_i' is correctly constructed, the EA provides a *NIZKP of selection* (Section 4.1).

This procedure effectively removes double votes, since only the last tuple from each credential group is retained. Moreover, ballot-stuffing tuples are rendered ineffective, as they do not have a corresponding "flag tuple".

Tally phase. The trustees take the resulting list of encrypted votes and apply a mixnet. Every vote is then threshold decrypted, and invalid votes (i.e., votes with decrypts to 1) are discarded. The remaining votes are tallied using standard techniques.

3.3 Design Decisions

Note that votes from *impersonation* and *ballot-stuffing* attacks are handled during the cleansing phase of the protocol, where NIZKPs are used to prove their correct treatment. An alternative approach would be to prevent such ballots in the submission phase already, by employing a stricter set of NIZKPs. For example, one could combine set-membership proofs with proofs of knowledge of discrete-logarithm representations, as in [18, 19]. However, these proofs are significantly more complex than those currently used in FiltrumVote and would substantially increase both proof size and verification time. Instead, we opt for NIZKPs that can be constructed largely from standard Σ-protocols, thereby simplifying the work of both the EA and independent auditors.

4 Detailed Specification

In this Section, we present a detailed description of FiltrumVote.

Notation. Let λ denote the security parameter. Let $\mathbb{G}_p$ be a cyclic group of prime order p with generator g, and h be an additional independently chosen generator of $\mathbb{G}_p$. We write $a \in_R A$ to denote that a is sampled uniformly at random from the set A. We denote by $H(\cdot)$ a cryptographic hash function.

4.1 Cryptographic building blocks

FiltrumVote uses cryptographic primitives commonly found in modern secure e-voting proposals, such as ElGamal encryption and verifiable shuffle protocols. The main exceptions are the specially designed NIZKPs. Nevertheless, these proofs are built using standard Σ-protocol techniques combined with the Fiat-Shamir heuristic. FiltrumVote also employs range proofs, which are less common in e-voting solutions but widely used in blockchain and other zero-knowledge-based protocols [23].

ElGamal public-key encryption scheme. The ElGamal encryption can be presented in two forms: the multiplicative variant $\mathcal{E} = (\text{KeyGen}, \text{Enc}, \text{Dec})$ and the exponential variant $\mathcal{E}^\star = (\text{KeyGen}^\star, \text{Enc}^\star, \text{Dec}^\star)$. In both key generation algorithms, $\text{KeyGen}(\lambda)$ and $\text{KeyGen}^\star(\lambda)$, have the same specification, i.e., sample a secret key $\mathsf{sk} \in_R \mathbb{Z}_p$, compute the public key $\mathsf{pk} = g^{\mathsf{sk}}$, and output $(\mathsf{sk}, \mathsf{pk})$.

In the multiplicative variant, a message $m \in \mathbb{G}_p$ is encrypted by sampling $r \in_R \mathbb{Z}_p$, and outputting the ciphertext $\mathsf{ct} = (a, b)$, with $a = g^r$ and $b = m \cdot \mathsf{pk}^r$. Decryption of (a, b), is performed by computing $m \leftarrow b \cdot a^{-sk}$.

In the exponential variant, a message $m \in \mathbb{Z}_p$ is encrypted by sampling $r \in_R \mathbb{Z}_p$ and outputting the ciphertext $\mathsf{ct} = (a, b)$, with $a = g^r$ and $b = g^m \cdot \mathsf{pk}^r$). Decryption of (a, b) proceeds by computing $m \leftarrow \log_g(b \cdot a^{-\mathsf{sk}})$. Since it requires solving a discrete logarithm, this form is practical only for small values of m.

Both exponential and multiplicative ElGamal ciphertexts (a, b) support re-randomization: sampling $r' \in_R \mathbb{Z}_p$, one computes $(a', b') = (a \cdot g^{r'}, b \cdot \mathsf{pk}^{r'})$. Moreover, the exponential variant is additively homomorphic, i.e., component-wise multiplication of two ciphertexts results in an encryption of the sum of the corresponding plaintexts

$$\mathrm{Enc}^\star(m_1, r_1) \cdot \mathrm{Enc}^\star(m_2, r_2) = \mathrm{Enc}^\star(m_1 + m_2, r_1 + r_2).$$

NIZKP proofs. Non-interactive zero-knowledge proofs (NIZKPs) are fundamental building blocks in cryptographic protocol design. They allow a prover to demonstrate knowledge of a witness w for a statement st without revealing any information about w itself. A common approach to constructing NIZKPs is to transform an interactive Σ-protocol into a non-interactive proof using the Fiat–Shamir heuristic. In this transformation, the prover first computes a commitment u using a random value r. Next, they derive the challenge by hashing the public parameters pp, statement st, and the commitment u. Finally, the prover computes the response using r and the witness w. Multiple $\Sigma-$protocols can be composed using logical connectives such as AND and OR.

In this section, we present the main NIZKPs employed in FiltrumVote. Each of them is obtained as a composition of simpler underlying Σ-protocols. Detailed descriptions and full constructions are provided in the full version of the paper.

NIZKP of possession of encrypted public-key. A proof of possession π_{poss} proves that a multiplicative ElGamal ciphertext (a, b) is an encryption of the voter's public key $\mathsf{pk}_i = g^{\mathsf{sk}_i}$, and the prover knows the corresponding secret key sk_i[1].

$$NIZKP[(r, sk_i) : a = g^r \wedge b = g^{sk_i} pk^r]$$

NIZKP of consistency. The proof of consistency π_{cons} takes two ciphertext pairs $[\![pk_1]\!], [\![\sigma_1]\!]$ and $[\![pk_2]\!], [\![\sigma_2]\!]$ and proves that their underlying plaintexts are either both equal or both different. Thus, the proof is a composition of two proofs of plaintext equality with two proofs of plaintext inequality. We assume that the prover knows the secret key sk required to decrypt these ciphertexts.

$$NIZKP[(sk) : (pk_1 = pk_2 \wedge \sigma_1 = \sigma_2) \vee (pk_1 \neq pk_2 \wedge \sigma_1 \neq \sigma_2)]$$

NIZKP of correct selection. The proof of selection π_{sel} receives as input two ciphertexts $[\![\sigma_1]\!]$ and $[\![\sigma_2]\!]$, a ciphertext $[\![f_2]\!]$, a ciphertext $[\![v]\!]_T$ and a reencrypted ciphertext V'. It proves that V' is either (1) a reencryption of $[\![v]\!]_T$ when $\sigma_1 = \sigma_2$

[1] Note that $(\mathsf{sk}_i, \mathsf{pk}_i)$ refers to the voter's key pair, whereas $(\mathsf{sk}, \mathsf{pk})$ refers to the Electoral Authority's key pair.

and $f_2 = 1$; or (2) an encryption of $[\![1]\!]_T$ otherwise. This mechanism is used in our system to ensure that only the most recent valid vote from each user is selected. The proof is a composition of a re-encryption proof, a plaintext-equality proof, a plaintext-inequality proof, and three proofs of plaintext-value constraints.

$$NIZKP[(sk, r) : (V' = ReEnc([\![v]\!]_T, r) \wedge \sigma_1 = \sigma_2 \wedge f_2 = 1) \\ \vee (V' = [\![1]\!]_T \wedge (\sigma_1 \neq \sigma_2 \vee f_2 = 0))]$$

NIZKP of greater-than. The proof of greater-than π_{gt} takes as input two ciphertext tuples $[\![\sigma_1]\!], [\![f_1]\!], [\![ctr_1]\!]$ and $[\![\sigma_2]\!], [\![f_2]\!], [\![ctr_2]\!]$ and proves that the second tuple is lexicographically greater than the first (i.e., $\sigma_2||f_2||ctr_2 > \sigma_1||f_1||ctr_1$). This proof is used in our system to sort encrypted votes. The proof is a composition of two range proofs, three plaintext-equality proofs, and two proofs of plaintext-value.

$$NIZKP[(sk) : (\sigma_2 > \sigma_1) \vee (\sigma_2 = \sigma_1 \wedge f_2 = 1 \wedge f_1 = 0) \\ \vee (\sigma_2 = \sigma_1 \wedge f_2 = f_1 \wedge ctr_2 > ctr_1)]$$

Range proofs. Range proofs prove that, given a commitment $V = g^r h^v$, the value v is in $[0, 2^\ell - 1]$. One classical technique for range proofs consists of simply decomposing v in ℓ bits, committing each bit, and proving that each commitment indeed commits to either 0 or 1. This results in a proof size linear to the number of bits l. Some optimizations are described in [10].

In our application, we use range proofs to compare two ciphertexts $[\![x_1]\!]$ and $[\![x_2]\!]$ and prove that $x_2 > x_1$. To do so, we make a homomorphic subtraction $[\![x]\!] \leftarrow [\![x_2]\!] \cdot [\![x_1]\!]^{-1}$, create a new commitment V with the same plaintext as $[\![x]\!]$, and prove that it commits to a positive value $x \in [0, 2^\ell - 1]$. This assumes that $x_1, x_2 \ll q$, which must be checked by the verifier.

Verifiable shuffle. A verifiable shuffle protocol takes a list of ciphertexts, re-encrypts them, and outputs a permuted list of ciphertexts. It is accompanied by a zero-knowledge proof of correctness that guarantees no ciphertext has been altered or replaced. FiltrumVote adopts the Wikström–Terelius protocol [26], which commits to a permutation matrix and proves in zero knowledge that this matrix was used in the shuffle. In addition, it is possible to shuffle not only a simple list of ciphertexts, but also a list of tuples of ciphertexts — one just needs to use the same permutation matrix across multiple verifiable-shuffle instances. This applies even if one position in a tuple uses a different encryption key from another position in the tuple.

4.2 FiltrumVote Protocol

In this section, we present a detailed description of FiltrumVote.

Setup phase. The EA determines the security parameter λ, and the length of the credential to support the number of voters n_v : $\ell \geq \lceil \log(n_v) \rceil$. It also generates an encryption key pair (pk, sk) and publishes the pk. In addition, the n_T election trustees jointly run a distributed key generation, obtaining and publishing an encryption key pk_T while keeping their private shares sk_i (for $i = 1, ..., n_T$). Finally, the EA defines the list of valid candidates, reserving the identifier 1 to represent an invalid vote.

Registration phase. Each voter vid_i samples a private key $sk_i \in_R \mathbb{G}_p$ and computes the public key $pk_i = g^{sk_i}$. The voter transmits pk_i over an authenticated channel to the EA. The EA then assigns the voter a unique, small credential $\sigma_i \in \{0,1\}^\ell$ and publishes the tuple (vid_i, pk_i, σ_i) in the voter roll R. Voters check that their public key pk_i is correctly registered in the voter roll.

Submission phase. To cast a vote v_i, the voter does the following:

- encrypts their vote, using the trustee's public key: $[\![v]\!]_T = Enc_{pk_T}(v)$;
- encrypts their public key, using EA's public key: $[\![pk_i]\!] = Enc_{pk}(pk_i)$;
- encrypts their credential, using EA's public key: $[\![\sigma_i]\!]^\star = Enc^\star_{pk}(\sigma_i)$;
  - Note that the credential is encrypted using exponential ElGamal. This provides an additive homomorphism, necessary to compare two ciphertexts using range-proofs (Section 4.1).
- computes a proof π_v that v encrypts a valid candidate[2];
- computes a *NIZKP of encrypted public-key possession* π_{poss} for $[\![pk_i]\!]$;
- computes a proof π_σ that $[\![\sigma_i]\!]^\star$ encrypts a small $\sigma \in [0, 2^\ell - 1]$ [3];
- sends the ballot $\beta_i = ([\![v_i]\!]_T, [\![pk_i]\!], [\![\sigma_i]\!]^\star, \pi_{kw}, \pi_{poss}, \pi_\sigma)$ to the EA through an authenticated channel.

The EA receives the ballot and checks if the proofs are valid and if there is no duplicate in the PBB. It also decrypts the credential and public key and checks if the pair (pk_i, σ_i) is present in the voter roll. If all checks succeed, the EA publishes the ballot on PBB. Voters can access PBB to check that their ballots were correctly registered.

Cleansing phase. Once the voting phase is finished, the EA starts the cleansing phase. It can be divided into five steps.

[2] This is necessary to avoid forced abstention attacks. For example, if v is an unlikely, arbitrary string, the presence of v in the decrypted votes could indicate whether the voter has revoted. Range proofs could be used, with the optimizations described in Section 5.2

[3] This is necessary since *NIZKP of greather-than* assumes $\sigma << p$. Again, range proofs can be used together with the optimizations described in Section 5.2.

Preparation. Let B be the list of all ballots in PBB, listed in chronological order, and let R be the voter roll. Then, the EA parses each element e_i of $B||R$ as a tuple $t_i = (\mathtt{v}_i, \mathtt{pk}_i, \boldsymbol{\sigma}_i, \mathtt{f}_i, \mathtt{ctr}_i)$, where

- If $e_i \in B$,

$$t_i \leftarrow ([\![v_i]\!]_T, [\![pk_i]\!], [\![\sigma_i]\!]^\star, Enc^\star_{pk}(0, r), Enc^\star_{pk}(ctr, r))$$

- Else if $e_i \in R$,

$$t_i \leftarrow (Enc_{pk_T}(1, r), Enc_{pk}(pk_i, r), Enc^\star_{pk}(\sigma_i, r), Enc^\star_{pk}(1, r), Enc^\star_{pk}(ctr, r))$$

Where ctr is an increasing counter and represents the chronological order of the votes, and $r = 0$ to facilitate verification from external auditors. Note that σ, ctr and f are encrypted using exponential ElGamal: σ and ctr for additive homomorphism, and f to simplify NIZKPs.

Mixing. The EA applies a verifiable shuffle in L_1. The resulting set L_2, and the proof of correct shuffle are published in PBB.

Sorting. The EA sorts the encrypted set L_2, producing L_3, as follows:

- It decrypts the credential, flag, and counter of each tuple in L_2: $(\sigma_i, f_i, ctr_i) = (Dec^\star(\boldsymbol{\sigma}_i), Dec^\star(\mathtt{f}_i), Dec^\star(\mathtt{ctr}_i))$. Since σ_i, f_i, and ctr_i are small numbers, this decryption is feasible. The EA keeps the resulting plaintexts private.
- It sorts the set according to the relation $t_i < t_j \iff \sigma_i||f_i||ctr_i < \sigma_j||f_j||ctr_j$, using any $O(n\log(n))$-complexity sorting algorithm, such as mergesort.
- It proves that the resulting set L_3 is correctly ordered by providing an *NIZKP of greater-than* π_{gt} for each pair of consecutive tuples.

The EA then publishes the sorted set L_3 together with the corresponding *greater-than* proofs.

Consistency Check. The EA proves that there are no impersonation-attack votes in L_3. To this end, it provides a *NIZKP of consistency* π_{cons} between each pair of consecutive votes. It then publishes the consistency proofs.

Selection. For each pair of tuple $t_i, t_{i+1} \in L_3$, the EA constructs a encrypted vote V'_i as follows:

- If t_i corresponds to a valid vote (i.e., $\sigma_i = \sigma_{i+1} \wedge f_{i+1} = 1$), $V'_i = ReEnc([\![v_i]\!]_T, r)$.
- If t_i corresponds to an invalid vote (i.e., $\sigma_i \neq \sigma_{i+1} \vee f_{i+1} = 0$), $V'_i = [\![1]\!]$.

The EA produces an *NIZKP of correct selection* for each V'_i. It then publishes the resulting set L_4 containing all V'_i, together with the corresponding selection proofs in PBB.

Tally Phase. The trustees take L_4 and apply a mixnet. Then, every vote V_i' is threshold decrypted. Votes whose decrypted plaintext is invalid (i.e., $v_i = 1$) are discarded. The remaining votes are tallied using standard techniques.

Verification Phase. Any participant, including the voter and external observer, can verify the correctness of the election by checking: (1) the voter roll correctness; (2) all *NIZKPs*; (3) the additional ciphertexts in the *preparation* step; and (4) the tally result.

5 Discussion

5.1 Security Analysis

In this section, we argue about the security properties of FiltrumVote. A formal security proof is left for future work.

Ballot secrecy. Ballot secrecy relies on a threshold number of honest trustees. Votes are encrypted under the trustees' public key; thus, if the final mixing is sound, the shuffling is privacy-preserving, and the ElGamal encryption is CPA-secure, it is infeasible to identify the voter's selection from public information.

Verifiability. Election verifiability does not rely on trusting the EA or the trustees; rather, it is guaranteed by the correctness of the NIZKPs. Because the PBB is append-only, the mixing protocol is sound, and vote tuples remain public at every step, any attempt by the EA to modify or remove data is detectable.

In addition, the protocol ensures that only the final vote cast by each eligible voter is counted. This is achieved through the cleansing phase: a preparation step that assigns a "flag tuple" to eligible voters; an ordering step that groups and chronologically sorts voters' tuples so the flag tuple appears last; and a selection step that isolates and counts only the votes immediately preceding the flag tuple. The integrity of each of these steps is publicly verifiable via NIZKPs.

Finally, no one, including the EA, can cast a valid vote without knowing an eligible voter's secret key. This is enforced by multiple verification checks in the pair (pk, σ). First, in the submission phase, the NIZKP of possession ensures only the secret key owner can send a ballot with the corresponding public key pk. Second, the consistency step rejects mismatched pairings like (pk', σ), where σ is valid but pk' is incorrect for that voter (Section 3.2). Finally, the selection step nullifies any tuple with invalid credentials (σ') by only accepting votes followed by valid flag tuples.

Coercion-resistance. Coercion resistance is ensured by an honest EA and a threshold number of honest trustees. This is achieved because all tuples on the PBB are encrypted, hiding the number of votes cast by any individual. Furthermore, the initial mixing ensures that the ordering process acts on what appears to be a random set, preventing any data leakage. Additionally, neither the NIZKPs

of consistency nor the NIZKPs of selection reveal voter-specific information. Finally, the final mixnet removes any link between the submitted vote tuples and the decrypted votes. Ultimately, the only publicly revealed information is the total number of cancelled votes, which does not offer any advantage to a coercer in a large enough election. Note that it does not even reveal if the voter has participated in an election, protecting against forced abstention attacks.

5.2 Performance Evaluation

To evaluate FiltrumVote's performance, we have implemented a proof-of-concept prototype[4] of the main NIZKP used in our solution: proof of possession π_{poss}, proof of greater-than π_{gt}, proof of consistency π_{cons}, and proof of selection π_{sel}. These are the only non-standard primitives in our solution. The prototype is implemented in Rust, and all operations are done over the elliptic curve P-256.

Experiments. All computations are done locally in a 12th Gen i7-12700H Intel Core CPU with 32GB RAM. We measured both proof generation and verification times for each NIZKP, averaging the results over 1000 repetitions. We use 24 bits to represent σ and ctr, which is more than enough to support an election with more than one million votes. Table 1 shows the results.

Since the range proofs in π_{gt} constitute the most computationally expensive component of the protocol, we introduce two optimizations. First, we reduce the bit-length of the credential σ_i and the counter ctr. Although our target remains one million votes, in practice, votes are tallied and published in batches (e.g., by geographic region). For example, one million votes can be partitioned into batches of 1024 voters, allowing σ_i to be represented with 10 bits. Furthermore, assuming a revoting rate of 0.5, a 12-bit counter ctr is sufficient. This alone reduces the proving time by almost half.

The second optimization we implement is fixed-base exponentiation precomputation. While all NIZKP proofs benefit from this technique, the impact is greater in π_{gt} due to the large number of exponentiations in the range proof. Specifically, we precompute a 16-bit multiplication table for both bases g and h. The table construction takes 350 ms, which can be performed only once before the election, and the resulting table requires 67 MB of storage.

Combining these two optimizations, we reduce the π_{gr} proof time from 26.64 ms to 2.89 ms, making it comparable to the simpler π_{cons} and π_{sel} proofs. Although more advanced techniques from [10] could be applied, they would introduce significant additional complexity while offering only marginal practical gains. In particular, even though such methods achieve asymptotically more efficient prover performance ($O(\log n)$), the small bit-lengths used in our range proofs limit the extent of these improvements in practice.

Analysis. We now estimate how long it takes to process 1 million votes in the main steps of the cleansing phase. If we consider a 0.5 rate of revoting and the

[4] https://github.com/no1-4now/FiltrumVote

Table 1: Average time for proving and verifying main NIZKPs and their proof size. The optimization opt.1 refers to a smaller number of bits for σ and ctr, while the optimization opt.2 refers to fixed-base exponentiation techniques.

NIZKP proof	prove (ms)	verify (ms)	proof size (bytes)
π_{poss}	0.33	0.53	96
π_{cons}	1.58	1.64	290
π_{sel}	2.14	2.36	321
π_{gt} (no-optimized)	26.64	26.68	6578
π_{gt} (opt.1)	15.68	13.77	3224
π_{gt} (opt. 2)	5.69	17.57	6578
π_{gt} (opt.1&2)	2.89	8.97	3224

additional flag tuples, we would need to process 2.5 million vote tuples. Considering the numbers in Table 1, this would require $\approx$ 1 hour in the consistency step, $\approx$ 1.5 hours in the selection step, and $\approx$ 2 hours in the ordering step to prove the NIZKP statements. Note that this estimate does not include other more standard tasks, such as encryption and decryption, and sorting tuples.

This results in a total of $\approx$ 4.5 hours to prove all NIZKPs to process 1 million votes. Note that this computation is highly parallelizable. For reference, DeVos voting can process only 80.000 votes in similar conditions — this considering a reduced, four-candidate election [24]. In addition, the NIZKPs for all 2.5 million tuples can be stored in only $\approx$ 9.15 GB. We therefore conclude that FiltrumVote is computationally feasible even for large-scale elections.

6 Conclusion and Future Works

We present FiltrumVote, a verifiable Internet voting system designed to resist coercion through deniable vote updating. The system is highly scalable, featuring a quasi-linear cleansing phase, and is capable of efficiently handling elections with up to one million votes. FiltrumVote relies primarily on zero-knowledge proofs using standard Σ-protocol techniques, enabling easier independent verification.

Future work includes developing formal security proofs to more rigorously assess FiltrumVote's security properties. Additionally, we plan to enhance the protocol to achieve quantum resistance, either by employing everlasting-privacy techniques [12] or by integrating post-quantum cryptographic primitives.

References

1. Achenbach, D., Kempka, C., Löwe, B., Müller-Quade, J.: Improved Coercion-Resistant electronic elections through deniable Re-Voting. USENIX JETS (2015)
2. Adida, B.: Helios: Web-based open-audit voting. In: USENIX security symposium (2008)
3. Aranha, D.F., Battagliola, M., Roy, L.: Faster coercion-resistant e-voting by encrypted sorting. In: EVOTE-ID (2023)

4. Araújo, R., Barki, A., Brunet, S., Traoré, J.: Remote electronic voting can be efficient, verifiable and coercion-resistant. In: Financial Cryptography (2016)
5. Araújo, R., Ben Rajeb, N., Robbana, R., Traoré, J., Youssfi, S.: Towards practical and secure coercion-resistant electronic elections. In: CANS (2010)
6. Cortier, V., Gaudry, P., Yang, Q.: Is the JCJ voting system really coercion-resistant? In: IEEE CSF (2024)
7. Cristiano, L., Longo, R., Spadafora, C.: Click and cast - assessing the usability of vote app. In: EVOTE-ID (2024)
8. Culnane, C., Schneider, S.: A peered bulletin board for robust use in verifiable voting systems. In: IEEE CSF (2014)
9. Cuvelier, É., Pereira, O., Peters, T.: Election verifiability or ballot privacy: Do we need to choose? In: ESORICS 2013 (2013)
10. Devillez, H., Pereira, O., Peters, T.: How to verifiably encrypt many bits for an election? In: ESORICS (2022)
11. Farzaliyev, V., Willemson, J.: End-to-end verifiable Internet voting with partially private bulletin boards. In: EVOTE-ID (2025)
12. Haines, T., Mueller, J., Mosaheb, R., Pryvalov, I.: Sok: Secure e-voting with everlasting privacy. In: Proceedings on Privacy Enhancing Technologies (PoPETs) (2023)
13. Haines, T., Müller, J., Querejeta-Azurmendi, I.n.: Scalable coercion-resistant e-voting under weaker trust assumptions. In: 38th ACM/SIGAPP SAC (2023)
14. Heiberg, S., Martens, T., Vinkel, P., Willemson, J.: Improving the verifiability of the Estonian Internet voting scheme. In: EVOTE-ID (2017)
15. Hirschi, L., Schmid, L., Basin, D.: Fixing the achilles heel of E-voting: the bulletin board. In: IEEE CSF (2021)
16. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: ACM WPES (2005)
17. Kiayias, A., Kuldmaa, A., Lipmaa, H., Siim, J., Zacharias, T.: On the security properties of e-voting bulletin boards. In: SCN (2018)
18. Locher, P., Haenni, R.: Verifiable Internet elections with everlasting privacy and minimal trust. In: VOTE-ID (2015)
19. Locher, P., Haenni, R., Koenig, R.E.: Coercion-resistant Internet voting with everlasting privacy. In: Financial Cryptography (2016)
20. Lueks, W., Querejeta-Azurmendi, I., Troncoso, C.: {VoteAgain}: A scalable coercion-resistant voting system. In: USENIX security (2020)
21. Marky, K., Zollinger, M.L., Roenne, P., Ryan, P.Y., Grube, T., Kunze, K.: Investigating usability and user experience of individually verifiable internet voting schemes. ACM-TOCHI (2021)
22. McMurtry, E., Pereira, O., Teague, V.: When is a test not a proof? In: ESORICS (2020)
23. Morais, E., Koens, T., Van Wijk, C., Koren, A.: A survey on zero knowledge range proofs and applications. SN Applied Sciences (2019)
24. Müller, J., Pejó, B., Pryvalov, I.: Devos: deniable yet verifiable vote updating. PoPETS (2024)
25. Nissen, C., Hilt, T., Budurushi, J., Volkamer, M., Kulyk, O.: Voting under pressure: Perceptions of counter-strategies in Internet voting. In: EVOTE-ID (2025)
26. Terelius, B., Wikström, D.: Proofs of restricted shuffles. In: AFRICACRYPT (2010)
27. Treier, T.: Re-voting under surveillance: National eID transaction logs as a threat to coercion resistance in Estonian Internet voting. In: EVOTE-ID (2025)

A Detailed description of NIZKPs

FiltrumVote makes extensive use of NIZKPs, composing them from several $\Sigma-$protocols. In this section, we provide a detailed description of all NIZKPs used in the solution.

A.1 Building blocks

NIZKP of plaintext value. Given an ElGamal ciphertext (a, b) in an exponential variant, we can prove that it encrypts a message $m = 0$:

$$NIZKP[(sk) : g^m = Dec^\star(a, b)] \iff NIZKP[(sk) : b = a^{sk} \wedge pk = g^{sk}]$$

Note that this is a standard proof of equality of discrete logarithms, for the statement $\text{st} = (a, b)$. The prover can samples $r' \in_R \mathbb{Z}_p$; compute the commitments $u_1 = g^{r'}$ and $u_2 = b^{r'}$, compute the challenge $c = H(\text{pp}, \text{st}, u)$ where $u = (u_1, u_2)$, and the response $z = r' - cr$. The verifier checks the proof $\pi_{dec} = (c, z)$ by computing $u'_1 = g^{r'}$ and $u'_2 = b^{r'}$, the challenge $c' = H(\text{pp}, \text{st}, u')$, and checking $c = c'$.

Similarly, we can prove that the ciphertext (a, b) encrypts the message $m = 1$. This is also standard proof of equality of discrete logarithms

$$NIZKP[(sk) : b/g = a^{sk} \wedge pk = g^{sk}]$$

NIZKP of correct decryption. We can prove that an ElGamal ciphertext (a, b) decrypts to a message m, through a standard proof of equality of discrete logarithms.

$$NIZKP[(sk) : pk = g^{sk} \wedge m = ba^{-sk}] \iff NIZKP[(sk) : pk = g^{sk} \wedge b/m = a^{sk})]$$

NIZKP of correct reencryption. We can prove that a given ElGamal ciphertext (a', b') is a correct reencryption of the ciphertext (a, b), through a standard proof of equality of discrete logarithms.

$$NIZKP[(r) : a' = a \cdot g^r \wedge b' = b \cdot pk^r] \iff NIZKP[(r) : a'/a = g^r \wedge b'/b = pk^r]$$

NIZKP of plaintext equality. We can prove that two ElGamal ciphertexts (a_1, b_1) and (a_2, b_2) encrypts the same message m. The prover generates a new ciphertext $(a', b') = (a_1/a_2, b_1/b_2)$ and shows that it encrypts the identity element ($m = 1$ in the multiplicative variant and $m = 0$ in the exponential variant). This results in a standard proof of equality of discrete logarithms.

$$NIZKP[(sk) : pk = g^{sk} \wedge b' = a'^{sk}]$$

NIZKP of plaintext inequality. We can prove that two ElGamal ciphertexts (a_1, b_1) and (a_2, b_2) do not encrypt the same message (i.e., $m_1 \neq m_2$). We use a technique similar to the *plaintext equivalent test* [22].

To produce the proof, the prover:

- computes a new ciphertext $(a', b') = (a_1/a_2, b_1/b_2)$.
- get a random exponent $\delta \in \mathbb{Z}_p^*$ and computes $m' = b'^{\delta} a'^{-\delta \cdot sk}$. Note that $m' = (m_1/m_2)^{\delta}$, which is different from 1 if and only if $m_1 \neq m_2$.
- proves the following (π):

$$NIZKP[(\delta, \delta \cdot sk) : 1 = pk^{\delta} g^{-\delta \cdot sk} \wedge m' = b'^{\delta} a'^{-\delta \cdot sk}]$$

Note that this is easy to prove with standard Sigma Protocol techniques.

To check the proof, the verifier:

- Computes the ciphertext $(a', b') = (a_1/a_2, b_1/b_2)$.
- Check the correctness of π.
- Check that $m' \neq 1$.

A.2 Main NIZKPs

NIZKP of encrypted public-key possession. We can prove that a multiplicative ElGamal ciphertext (a, b) encrypts a public-key $pk_i = g^{sk_i}$, and that the prover knows the corresponding sk_i.

$$NIZKP[(r, sk_i) : a = g^r \wedge b = g^{sk_i} pk^r]$$

Note that this proof π_{poss} can be done using traditional $\Sigma-$protocol techniques. The prover samples $s_1, s_2 \in \mathbb{Z}_p$, and computes the commitments $u = (g^{s_1}, g^{s_2} pk^{s_1})$, the challenge $c = H(y, u)$, and the responses $z_1 = s_1 - cr$ and $z_2 = s_2 - c \cdot sk_i$. The verifier checks the proof $\pi_{poss} = (c, z_1, z_2)$ by computing $u_1 = a^c \cdot g^{z_1}$ and $u_2 = b^c \cdot g^{z_2} \cdot pk^{z_1}$, computing $c' = H(y, u)$, and checking $c' = c$.

NIZKP of consistency. The proof of consistency π_{cons} takes two ciphertext pairs $[\![pk_1]\!], [\![\sigma_1]\!]^{\star}$ and $[\![pk_2]\!], [\![\sigma_2]\!]^{\star}$ and proves that their underlying plaintexts are either both equal or both different. We assume that the prover knows the secret key sk required to decrypt these ciphertexts. Essentially, the proof establishes:

$$NIZKP[(sk) : (pk_1 = pk_2 \wedge \sigma_1 = \sigma_2) \vee (pk_1 \neq pk_2 \wedge \sigma_1 \neq \sigma_2)]$$

Note that this proof π_{cons} is a composition of two proofs of plaintext equality and two proofs of plaintext inequality. The prover does the following:

- Consider $[\![pk_1]\!], [\![pk_2]\!], [\![\sigma_1]\!]^{\star}, [\![\sigma_2]\!]^{\star}$ as $(a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4)$
- Compute the ciphertexts $(\hat{a}, \hat{b}) = (a_1/a_2, b_1/b_2)$ and $(\tilde{a}, \tilde{b}) = (a_3/a_4, b_3/b_4)$
- If both plaintexts are equal, define random $\hat{m}, \tilde{m} \in_R \mathbb{G}_p$

– If both plaintexts are different, sample random $\hat{\delta}, \tilde{\delta} \in_R \mathbb{Z}_p$ and define $\hat{m} = (pk_1/pk_2)^{\hat{\delta}}$ and $\tilde{m} = g^{(\sigma_1-\sigma_2)\tilde{\delta}}$
– Prove the following:

$$\begin{aligned}\text{NIZKP}[(sk, \hat{\delta}, \hat{\delta}\cdot sk, \tilde{\delta}, \tilde{\delta}\cdot sk): \quad & (pk = g^{sk} \wedge \hat{b} = \hat{a}^{sk} \wedge \tilde{b} = \tilde{a}^{sk}) \\ \vee\, & (1 = pk^{\hat{\delta}} g^{-\hat{\delta}\cdot sk} \wedge \hat{m} = \hat{b}^{\hat{\delta}} \hat{a}^{-\hat{\delta}\cdot sk} \wedge 1 = pk^{\tilde{\delta}} g^{-\tilde{\delta}\cdot sk} \wedge \tilde{m} = \tilde{b}^{\tilde{\delta}} \tilde{a}^{-\tilde{\delta}\cdot sk})]\end{aligned}$$

Figure 2 shows the exact protocol to prove the relationship. The prover then sends $\hat{m}, \tilde{m}$ and the relationship proof to the verifier. To check the consistency proof, the verifier:

- Compute the ciphertexts $(\hat{a}, \hat{b}) = (a_1/a_2, b_1/b_2)$ and $(\tilde{a}, \tilde{b}) = (a_3/a_4, b_3/b_4)$
- Check the relationship proof as shown in Figure 2
- Check that $\hat{m}, \tilde{m} \neq 1$

$w = sk$	$w = (\hat{\delta}, \hat{\delta}\cdot sk, \tilde{\delta}, \tilde{\delta}\cdot sk)$
$c_2, s_1, z_2, z_3, z_4, z_5 \leftarrow \mathbb{Z}_p^*$	$c_1, z_1, s_2, s_3, s_4, s_5 \leftarrow \mathbb{Z}_p^*$
$u_1 = g^{s_1}$	$u_1 = g^{z_1} pk^{-c_1}$
$u_2 = \hat{a}^{s_1}$	$u_2 = \hat{a}^{z_1} \hat{b}^{-c_1}$
$u_3 = \tilde{a}^{s_1}$	$u_3 = \tilde{a}^{z_1} \tilde{b}^{-c_1}$
$u_4 = pk^{z_2} g^{-z_3}$	$u_4 = pk^{s_2} g^{-s_3}$
$u_5 = \hat{b}^{z_2} \hat{a}^{-z_3} \hat{m}^{-c_2}$	$u_5 = \hat{b}^{s_2} \hat{a}^{-s_3}$
$u_6 = pk^{z_4} g^{-z_5}$	$u_6 = pk^{s_4} g^{-s_5}$
$u_7 = \tilde{b}^{z_4} \tilde{a}^{-z_5} \tilde{m}^{-c_2}$	$u_7 = \tilde{b}^{s_4} \tilde{a}^{-s_5}$
$c = H(\text{pp}, \text{st}, u_1, \ldots, u_7)$	
$c_1 = c - c_2$	$c_2 = c - c_1$
$z_1 = s_1 + c_1 sk$	$z_2 = s_2 + c_2\hat{\delta}$
	$z_3 = s_3 + c_2\hat{\delta}\cdot sk$
	$z_4 = s_4 + c_2\tilde{\delta}$
	$z_5 = s_5 + c_2\tilde{\delta}\cdot sk$
Sends $\pi = (c_1, c_2, z_1, \ldots, z_5)$	

Receives $\pi = (c_1, c_2, z_1, \ldots, z_5)$
$u_1' \leftarrow g^{z_1} pk^{-c_1}$
$u_2' \leftarrow \hat{a}^{z_1} \hat{b}^{-c_1}$
$u_3' \leftarrow \tilde{a}^{z_1} \tilde{b}^{-c_1}$
$u_4' \leftarrow pk^{z_2} g^{-z_3}$
$u_5' \leftarrow \hat{b}^{z_2} \hat{a}^{-z_3} \hat{m}^{-c_2}$
$u_6' \leftarrow pk^{z_4} g^{-z_5}$
$u_7' \leftarrow \tilde{b}^{z_4} \tilde{a}^{-z_5} \tilde{m}^{-c_2}$
$c_1 + c_2 \stackrel{?}{=} H(\text{pp}, \text{st}, u_1', \ldots, u_7')$

Fig. 2: NIZKP of consistency protocol. (Left) Prover steps for generating the proof using witness $w = sk$ or $w = (\hat{\delta}, \hat{\delta}\cdot sk, \tilde{\delta}, \tilde{\delta}\cdot sk)$. (Right) Verifier steps.

NIZKP of correct selection. We can receive two ciphertexts $[\![\sigma_1]\!]^\star$ and $[\![\sigma_2]\!]^\star$, a ciphertext $[\![f_2]\!]$, a ciphertext $[\![v]\!]_T$, and prove that a reencrypted ciphertext V' is either (1) a reencryption of $[\![v]\!]_T$, if $\sigma_1 = \sigma_2$ and $f_2 = 1$; or (2) an encryption of $[\![1]\!]_T$ otherwise:

$$\begin{aligned}NIZKP[(sk, r) :& (V' = ReEnc([\![v]\!]_T, r) \wedge \sigma_1 = \sigma_2 \wedge f_2 = 1) \\ & \vee (V' = [\![1]\!]_T \wedge (\sigma_1 \neq \sigma_2 \vee f_2 = 0))]\end{aligned}$$

Note that this is a composition between a reencryption proof, a plaintext equality proof, a plaintext inequality proof, and three proofs of plaintext value. The prover does the following:

- Consider $V', [\![v]\!]_T, [\![\sigma_1]\!]^\star, [\![\sigma_2]\!]^\star, [\![f_2]\!]$ as tuples $(a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4), (a_5, b_5)$.
  - Recall that V' and $[\![v]\!]_T$ are encrypted under the trustee's public key pk_T, while $[\![\sigma_1]\!]^\star, [\![\sigma_2]\!]^\star, [\![f_2]\!]$ are encrypted under EA public key pk.
- Computes the ciphertexts $(\tilde{a}, \tilde{b}) = (a_1/a_2, b_1/b_2)$ and $(\hat{a}, \hat{b}) = (a_3/a_4, b_3/b_4)$
- If $\sigma_1 = \sigma_2$, defines random $\hat{m} \in_R \mathbb{G}_p$
- If $\sigma_1 \neq \sigma_2$, samples random $\delta \in_R \mathbb{Z}_p$ and defines $\hat{m} = g^{(\sigma_1 - \sigma_2)\delta}$
- Proves the following:

$$
\begin{aligned}
&\mathrm{NIZKP}\Big[(sk, r, r', \delta, \delta \cdot sk) : \\
&\quad (\tilde{a} = g^r \wedge \tilde{b} = pk_T^r \wedge pk = g^{sk} \wedge \hat{b} = \hat{a}^{sk} \wedge b_5/g = a_5^{sk}) \\
&\quad \vee \Big(a_1 = g^{r'} \wedge b_1/g = pk_T^{r'} \\
&\qquad \wedge \Big((1 = pk^\delta g^{-\delta \cdot sk} \wedge \hat{m} = \hat{b}^\delta \hat{a}^{-\delta \cdot sk}) \vee (pk = g^{sk} \wedge b_5 = a_5^{sk}) \Big) \Big) \Big]
\end{aligned}
$$

$w = (sk, r)$	$w = r'$	
	$w = (\delta, \delta \cdot sk)$	$w = sk$
$c_2, c_3, s_1, s_2,$ $z_3, z_4, z_5, z_6 \leftarrow \mathbb{Z}_p^*$	$c_1, s_3, z_1, z_2 \leftarrow \mathbb{Z}_p^*$	
$u_1 = g^{s_1}$	$u_1 = g^{z_1} \tilde{a}^{-c_1}$	
$u_2 = pk_T^{s_1}$	$u_2 = pk_T^{z_1} \tilde{b}^{-c_1}$	
$u_3 = g^{s_2}$	$u_3 = g^{z_2} pk^{-c_1}$	
$u_4 = \hat{a}^{s_2}$	$u_4 = \hat{a}^{z_2} \hat{b}^{-c_1}$	
$u_5 = a_5^{s_2}$	$u_5 = a_5^{z_2} (b_5/g)^{c_1}$	
$u_6 = g^{z_3} a_1^{-c_2}$	$u_6 = g^{s_3}$	
$u_7 = pk_T^{z_3} (b_1/g)^{-c_2}$	$u_7 = pk_T^{s_3}$	
$c_4 = c_2 - c_3$	$c_4, s_4, s_5, z_6 \leftarrow \mathbb{Z}_p^*$	$c_3, s_6, z_4, z_5 \leftarrow \mathbb{Z}_p^*$
$u_8 = pk^{z_4} g^{-z_5}$	$u_8 = pk^{s_4} g^{-s_5}$	$u_8 = pk^{z_4} g^{-z_5}$
$u_9 = \hat{b}^{z_4} \hat{a}^{-z_5} \hat{m}^{-c_3}$	$u_9 = \hat{b}^{s_4} \hat{a}^{-s_5}$	$u_9 = \hat{b}^{z_4} \hat{a}^{-z_5} \hat{m}^{-c_3}$
$u_{10} = g^{z_6} pk^{-c_4}$	$u_{10} = g^{z_6} pk^{-c_4}$	$u_{10} = g^{s_6}$
$u_{11} = a_5^{z_6} b_5^{-c_4}$	$u_{11} = a_5^{z_6} b_5^{-c_4}$	$u_{11} = a_5^{s_6}$
$c = H(\mathrm{pp}, \mathrm{st}, u_1, \ldots, u_{11})$		
$c_1 = c - c_2$	$c_2 = c - c_1$	
$z_1 = s_1 + c_1 r$	$z_3 = s_3 + c_2 r'$	
$z_2 = s_2 + c_1 sk$	$c_3 = c_2 - c_4$	$c_4 = c_2 - c_3$
	$z_4 = s_4 + c_3 \delta$	$z_6 = s_6 + c_4 sk$
	$z_5 = s_5 + c_3 \delta \cdot sk$	
Sends $\pi = (c_1, c_3, c_4, z_1, \ldots, z_6)$		

Receives $\pi =$ $(c_1, c_3, c_4, z_1, \ldots, z_6)$
$c_2 \leftarrow c_3 + c_4$
$u'_1 \leftarrow g^{z_1} \tilde{a}^{-c_1}$
$u'_2 \leftarrow pk_T^{z_1} \tilde{b}^{-c_1}$
$u'_3 \leftarrow g^{z_2} pk^{-c_1}$
$u'_4 \leftarrow \hat{a}^{z_2} \hat{b}^{-c_1}$
$u'_5 \leftarrow a_5^{z_2} (b_5/g)^{c_1}$
$u'_6 \leftarrow g^{z_3} a_1^{-c_2}$
$u'_7 \leftarrow pk_T^{z_3} (b_1/g)^{-c_2}$
$u'_8 \leftarrow pk^{z_4} g^{-z_5}$
$u'_9 \leftarrow \hat{b}^{z_4} \hat{a}^{-z_5} \hat{m}^{-c_3}$
$u'_{10} \leftarrow g^{z_6} pk^{-c_4}$
$u'_{11} \leftarrow a_5^{z_6} b_5^{-c_4}$
$c_1 + c_2 + c_3 + c_4 \stackrel{?}{=} H(g, pk, u'_1, \ldots, u'_{11})$

Fig. 3: NIZKP of correct selection protocol. (Left) Prover steps for generating the proof using witness $w = (sk, r)$, $w = (r', \hat{\delta}, \hat{\delta} \cdot sk)$ or $w = (r', sk)$. (Right) Verifier steps for verifying the proof.

Figure 3 shows the exact protocol to prove the relationship. The prover then sends $\hat{m}$ and the relationship proof to the verifier. To check the correct selection proof, the verifier:

- Computes the ciphertexts $(\tilde{a}, \tilde{b}) = (a_1/a_2, b_1/b_2)$ and $(\hat{a}, \hat{b}) = (a_3/a_4, b_3/b_4)$
- Checks the relationship proof as shown in Figure 2
- Checks that $\hat{m} \neq 1$

NIZKP of greater-than. We can prove that the ciphertext tuple $[\![\sigma_2]\!]^\star, [\![f_2]\!], [\![ctr_2]\!]^\star$ is lexicographically greater than $[\![\sigma_1]\!]^\star, [\![f_1]\!], [\![ctr_1]\!]^\star$ (i.e., $\sigma_2||f_2||ctr_2 > \sigma_1||f_1||ctr_1$).

$$\begin{aligned} NIZKP[(sk) :& (\sigma_2 > \sigma_1) \\ & \vee (\sigma_2 = \sigma_1 \wedge f_2 = 1 \wedge f_1 = 0) \\ & \vee (\sigma_2 = \sigma_1 \wedge f_2 = f_1 \wedge ctr_2 > ctr_1)] \end{aligned}$$

Note that this is a composition of two range proofs, three plaintext equality proofs, and two proofs about plaintext value. The prover does the following:

- Consider $[\![\sigma_1]\!]^\star, [\![\sigma_2]\!]^\star, [\![f_1]\!], [\![f_2]\!], [\![ctr_1]\!]^\star, [\![ctr_2]\!]^\star$ as tuples $(a_1, b_1), (a_2, b_2), (a_3, b_3), (a_4, b_4), (a_5, b_5), (a_6, b_6)$.
- Computes the ciphertexts $(a_\sigma, b_\sigma) = (a_2/a_1, b_2/b_1 g)$,$(a_f, b_f) = (a_4/a_3, b_4/b_3)$, and $(a_{ctr}, b_{ctr}) = (a_6/a_5, b_6/b_5 g)$
  - Consider that they are encryption of $\sigma = \sigma_2 - \sigma_1 - 1$, $f = f_2 - f_1$, and $ctr = ctr_2 - ctr_1 - 1$
- Compute the commitments $V_\sigma = g^{r_\sigma} h^\sigma$, and $V_{ctr} = g^{r_{ctr}} h^{ctr}$
- Compute the binary decomposition of σ. Samples $r_\sigma^{(i)} \in_R \mathbb{Z}_p$ for $i \in [0, \ell - 1]$, such that $r_\sigma = r_\sigma^{(0)} + 2r_\sigma^{(1)} + ... + 2^{\ell-1} r_\sigma^{(\ell-1)}$. Compute the commitment of each bit.

$$\sigma = \sigma^{(0)} + 2\sigma^{(1)} + ... + 2^{l-1}\sigma^{(l-1)}, \text{ where } \sigma^{(i)} \in \{0, 1\}$$
$$com_\sigma^{(i)} = g^{r_\sigma^{(i)}} h^{\sigma^{(i)}}, \text{ for } i \in [0, l-1]$$

- Compute the binary decomposition of ctr. Samples $r_{ctr}^{(i)} \in_R \mathbb{Z}_p$ for $i \in [0, \ell - 1]$, such that $r_{ctr} = r_{ctr}^{(0)} + 2r_{ctr}^{(1)} + ... + 2^{\ell-1} r_{ctr}^{(\ell-1)}$. Compute the commitment of each bit.

$$ctr = ctr^{(0)} + 2ctr^{(1)} + ... + 2^{l-1}ctr^{(l-1)}, \text{ where } ctr^{(i)} \in \{0, 1\}$$
$$com_{ctr}^{(i)} = g^{r_{ctr}^{(i)}} h^{ctr^{(i)}}, \text{ for } i \in [0, l-1]$$

- Proves the following:

$$
\begin{aligned}
\mathrm{NIZKP}\Bigg[& (sk, r_\sigma, \sigma, r_\sigma^{(1)}, \ldots, r_\sigma^{(\ell-1)}, r_{ctr}, ctr, r_{ctr}^{(1)}, \ldots, r_{ctr}^{(\ell-1)}) : \\
& (pk = g^{sk} \wedge V_\sigma = g^{r_\sigma} h^\sigma \wedge b_\sigma = g^\sigma a_\sigma^{sk} \\
& \qquad \wedge_{i=0}^{\ell-1} \left(com_\sigma^{(i)} = g^{r_\sigma^{(i)}} \vee com_\sigma^{(i)} / h = g^{r_\sigma^{(i)}} \right)) \\
& \vee (pk = g^{sk} \wedge b_\sigma = a_\sigma^{sk} \wedge b_4 / g = a_4^{sk} \wedge b_3 = a_3^{sk}) \\
& \vee (pk = g^{sk} \wedge b_\sigma = a_\sigma^{sk} \wedge b_f = a_f^{sk} \wedge V_{ctr} = g^{r_{ctr}} h^{ctr} \wedge b_{ctr} = g^{ctr} a_{ctr}^{sk} \\
& \qquad \wedge_{i=0}^{\ell-1} \left(com_{ctr}^{(i)} = g^{r_{ctr}^{(i)}} \vee com_{ctr}^{(i)} / h = g^{r_{ctr}^{(i)}} \right)) \Bigg]
\end{aligned}
$$

$w = (sk, r_\sigma, \sigma)$	$w = sk$	$w = (sk, r_{ctr}, ctr)$
$s_1, s_2, s_3 \leftarrow \mathbb{Z}_p^*$	$z_1, z_2, z_3, c_1 \leftarrow \mathbb{Z}_p^*$	
$u_\sigma, ctx_\sigma = \mathrm{Range.com}(\sigma, com_\sigma^{(i)})$	$u_\sigma, z_\sigma, c_\sigma = \mathrm{Range.sim}(c_1, com_\sigma^{(i)})$	
$u_1 = g^{s_1}$	$u_1 = g^{z_1} pk^{-c_1}$	
$u_2 = g^{s_3} h^{s_2}$	$u_2 = g^{z_3} h^{z_2} V_\sigma^{-c_1}$	
$u_3 = g^{s_2} a_\sigma^{s_1}$	$u_3 = g^{z_2} a_\sigma^{z_1} b_\sigma^{-c_1}$	
$z_4, c_2 \leftarrow \mathbb{Z}_p^*$	$s_4 \leftarrow \mathbb{Z}_p^*$	$z_4, c_2 \leftarrow \mathbb{Z}_p^*$
$u_4 = g^{z_4} pk^{-c_2}$	$u_4 = g^{s_4}$	$u_4 = g^{z_4} pk^{-c_2}$
$u_5 = a_\sigma^{z_4} b_\sigma^{-c_2}$	$u_5 = a_\sigma^{s_4}$	$u_5 = a_\sigma^{z_4} b_\sigma^{-c_2}$
$u_6 = a_4^{z_4} (b_4/g)^{-c_2}$	$u_6 = a_4^{s_4}$	$u_6 = a_4^{z_4} (b_4/g)^{-c_2}$
$u_7 = a_3^{z_4} b_3^{-c_2}$	$u_7 = a_3^{s_4}$	$u_7 = a_3^{z_4} b_3^{-c_2}$
$z_5, z_6, z_7, c_3 \leftarrow \mathbb{Z}_p^*$		$s_5, s_6, s_7 \leftarrow \mathbb{Z}_p^*$
$u_8 = g^{z_5} pk^{-c_3}$		$u_8 = g^{s_5}$
$u_9 = a_\sigma^{z_5} b_\sigma^{-c_3}$		$u_9 = a_\sigma^{s_5}$
$u_{10} = a_f^{z_5} b_f^{-c_3}$		$u_{10} = a_f^{s_5}$
$u_{11} = g^{z_7} h^{z_6} V_{ctr}^{-c_3}$		$u_{11} = g^{s_7} h^{s_6}$
$u_{12} = g^{z_6} a_{ctr}^{z_5} b_{ctr}^{-c_3}$		$u_{12} = g^{s_6} a_{ctr}^{s_5}$
$u_{ctr}, z_{ctr}, c_{ctr} = \mathrm{Range.sim}(c_3, com_{ctr}^{(i)})$		$u_{ctr}, ctx_{ctr} = \mathrm{Range.com}(ctr, com_{ctr}^{(i)})$
$c = H(\mathrm{pp}, \mathrm{st}, u_1, \ldots, u_{12}, u_\sigma, u_{ctr})$		
$c_1 = c - c_2 - c_3$	$c_2 = c - c_1 - c_3$	$c_3 = c - c_1 - c_2$
$z_1 = s_1 + c_1 sk$	$z_4 = s_4 + c_2 sk$	$z_5 = s_5 + c_3 sk$
$z_2 = s_2 + c_1 \sigma$		$z_6 = s_6 + c_3 ctr$
$z_3 = s_3 + c_1 r_\sigma$		$z_7 = s_7 + c_3 r_{ctr}$
$z_\sigma, c_\sigma = \mathrm{Range.resp}(ctx_\sigma, c_1, \sigma, r_\sigma)$		$z_{ctr}, c_{ctr} = \mathrm{Range.resp}(ctx_{ctr}, c_3, ctr, r_{ctr})$
Sends $\pi = (c_1, c_2, c_3, c_\sigma, c_{ctr}, z_1, \ldots, z_7, z_\sigma, z_{ctr})$		

Fig. 4: NIZKP of greater-than proof. Prover steps for generating the proof using witness $w = (sk, r)$, $w = (r', \hat{\delta}, \hat{\delta} \cdot sk)$ or $w = (r', sk)$. Details of subfunctions Range are in Figure 6

Figure 4 shows the exact protocol to prove the relationship. The prover then sends $V_\sigma, V_{ctr}, com_\sigma^{(i)}, com_{ctr}^{(i)}$ and the relationship proof to the verifier. To check the greater-than proof, the verifier:

- Computes the ciphertexts $(a_\sigma, b_\sigma) = (a_2/a_1, b_2/b_1 g)$,$(a_f, b_f) = (a_4/a_3, b_4/b_3)$, and $(a_{ctr}, b_{ctr}) = (a_6/a_5, b_6/b_5 g)$
- Checks the relationship proof as shown in Figure 5
- Check the following:

$$V_\sigma \stackrel{?}{=} \prod_{i=0}^{\ell-1} (com_\sigma^{(i)})^{2^i} \text{ and } V_{ctr} \stackrel{?}{=} \prod_{i=0}^{\ell-1} (com_{ctr}^{(i)})^{2^i}$$

Receives $\pi = (c_1, c_2, c_3, c_\sigma, c_{ctr}, z_1, \ldots, z_7, z_\sigma, z_{ctr})$
$u'_\sigma \leftarrow \text{Range.ver}(com_\sigma^{(i)}, c_1, c_\sigma, z_\sigma)$
$u'_{ctr} \leftarrow \text{Range.ver}(com_{ctr}^{(i)}, c_3, c_{ctr}, z_{ctr})$
$u'_1 \leftarrow g^{z_1} pk^{-c_1}$
$u'_2 \leftarrow g^{z_2} h^{z_3} V_\sigma^{-c_1}$
$u'_3 \leftarrow g^{z_2} a_\sigma^{z_1} b_\sigma^{-c_1}$
$u'_4 \leftarrow g^{z_4} pk^{-c_2}$
$u'_5 \leftarrow a_\sigma^{z_4} b^{-c_2}$
$u'_6 \leftarrow a_4^{z_3} (b_4/g)^{-c_2}$
$u'_7 \leftarrow a_3^{z_3} b_3^{-c_2}$
$u'_8 \leftarrow g^{z_5} pk^{-c_3}$
$u'_9 \leftarrow a_\sigma^{z_5} b_\sigma^{-c_3}$
$u'_{10} \leftarrow a_f^{z_5} b_f^{-c_3}$
$u'_{11} \leftarrow g^{z_7} h^{z_6} V_{ctr}^{-c_3}$
$u'_{12} \leftarrow g^{z_6} a_{ctr}^{z_5} b_{ctr}^{-c_3}$
$c_1 + c_2 + c_3 \stackrel{?}{=} H(\text{pp}, \text{st}, u_1, \ldots, u_{12}, u_\sigma, u_{ctr}))$

Fig. 5: NIZKP of greater-than proof. Verifier steps

B Detailed description of Range Proofs

Range proofs using $\Sigma-$protocols We can prove that a commitment $V = g^r h^v$ commits to a value $v \in [0, 2^\ell - 1]$ as follows:

- Obtain a binary decomposition of v

$$v = v_0 + 2v_1 + 2^2 v_2 + ... + 2^{\ell-1} v_{\ell-1} \text{ , where } v_i \in \{0, 1\}$$

- Samples $r_i \in_R \mathbb{Z}_p$ for $i \in [0, \ell-1]$, such that $r = r_0 + 2r_1 + 2^2 r_2 + ... + 2^{\ell-1} r_{\ell-1}$
- Compute the commitments $com_i = g^{r_i} h^{v_i}$ for $i \in [0, \ell - 1]$
- Prove that every commitment com_i commits to either 0 or 1, i.e.,

$$NIZKP[(r_0, ..., r_{\ell-1}) : \bigwedge_{i=0}^{\ell-1} (com_i = g^{r_i} \vee com_i = g^{r_i} h)]$$

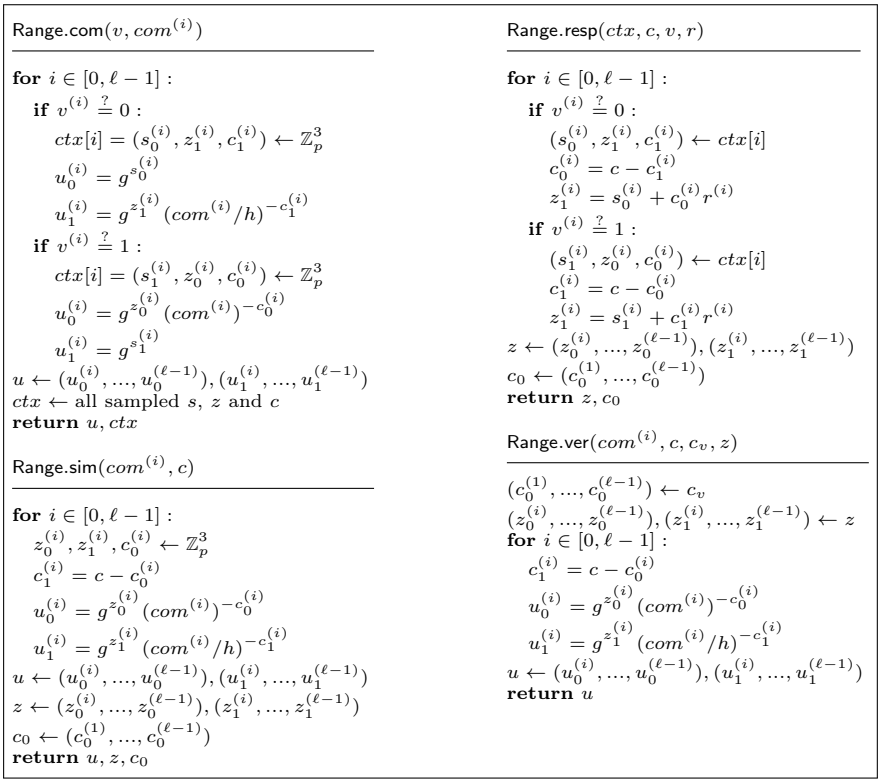

Fig. 6: Details of subfunctions for proving 0-1, used in range proofs

– Output the commitments and the 0-or-1 NIZKP proofs

The verification procedure is as follows

– Verify the 0-or-1 NIZKPs proofs
– Check the following:

$$V = \prod_{i=0}^{\ell-1} (com_i)^{2^i}$$

Without any optimizations, computing the proof requires 4ℓ exponentiations, and verifying also requires 4ℓ exponentiations. The proof size is $3\ell + 1$ elements in $\mathbb{Z}_p$ and ℓ elements in $\mathbb{G}_p$

Comparing two ciphertexts. To prove that $\sigma_2 > \sigma_1$ given only their ciphertexts, we first make a homomorphic subtraction, homomorphically subtract 1 (to remove cases when $\sigma_2 = \sigma_1$), and then prove that the resulting value encrypts a non-negative value. That is, supposing exponential ElGamal ciphertexts (a_1, b_1) and (a_2, b_2), and $\sigma_i < 2^\ell \ll p$, the procedure is as follows:

- Generate a new ciphertext $(a', b') = (a_2/a_1, b_2/b_1 g)$
- Decrypt the ciphertext (a', b'), obtaining σ'
- Computes a NIZKP that (a', b') and V has the same plaintext
- Creates a rangeproof that V has a value $v \in [0, 2^\ell - 1]$

To prove that (a', b') and V have the same plaintext, we prove the following relationship:

$$NIZKP[(sk, \sigma, r) : V = g^r h^\sigma \wedge g^\sigma = ba^{-sk} \wedge pk = g^{sk}]$$
$$\iff NIZKP[(sk, \sigma, r) : V = g^r h^\sigma \wedge b = g^\sigma a^{sk} \wedge pk = g^{sk}]$$

This can be proved using standard $\Sigma-$protocol techniques.

C Everlasting Privacy in FiltrumVote

There are essentially two main approaches to achieving everlasting privacy in voting systems: anonymizing each ballot in PBB (which [12] calls B-ANON) and using information-theoretical private-preserving techniques to tally ballots (which [12] calls B-ID).

C.1 B-ANON

In FiltrumVote, the B-ANON approach could be achieved by substituting the encryption of the voter credential σ_i and the public key σ_i for commitments. Thus, the public ballot would be $\beta_i = (\llbracket v_i \rrbracket_T, \langle pk_i \rangle, \langle \sigma_i \rangle)$, where $\llbracket x \rrbracket_T$ is the ElGamal encryption with the Trustees' public key and $\langle x \rangle$ is the perfectly hiding Pedersen Commitment. Of course, to enable the EA to recover pk_i and σ_i, the voter is also required to submit the corresponding openings $\hat{\beta} = ((pk_i, r_i), (\sigma_i, s_i)$ privately to EA. Since Pedersen Commitments are structurally similar to ElGamal ciphertexts, the EA is able to produce all NIZKPs with little modification.

However, the ordering phase still reveals partial information about σ, as tuples with smaller σ will always come before tuples with bigger σ. So, it is also necessary to hide which credential σ is assigned to which voter. To do so in a verifiable manner, one approach is to change the registration phase. Before the election, the EA first publicly generates n_v unique credentials and commits them, generating $\langle \sigma_i \rangle$. Then, it shuffles these committed credentials through a mixing protocol. The EA then assigns one of these committed credentials to each voter, publishing the tuple $(vid, pk_i, \langle \sigma_i \rangle)$ to the voter roll. Each voter vid receives from the EA the opening for her assigned commitment $\langle \sigma_i \rangle$. Figure 7 illustrates this flow. In addition, voters should make a new commitment for σ_i when casting a vote, so that the public ballot is $\beta_i = (\llbracket v_i \rrbracket_T, \langle pk_i \rangle, \langle \sigma_i \rangle')$.

However, B-ANON approaches rely on a strong assumption: ballots must be cast through an anonymous channel. Achieving this in practice is challenging, as any side information may reveal the identity of the ballot owner. For example, an attacker who obtains a voter's tracking code (e.g., through deception or coercion)

can identify the corresponding ballot. Moreover, if the attacker knows at which time a target voter intends to cast their ballot, they can simply monitor the PBB during that period and identify the ballot submitted at that time. Note that it is not possible to "hide" the PBB during vote submission, since revoting schemes inherently rely on the order and timing in which ballots are cast.

In addition, in FiltrumVote, the B-ANON approach does not protect cancelled votes nor hide revoting patterns. Therefore, if an attacker can decrypt all ciphertexts (e.g., due to quantum computers), he is able to see which candidates the cancelled votes contain, how many votes each (anonymous) voter has cast, among other things. Although none of these directly breaks vote privacy, they still reveal more information than the EA might want to reveal.

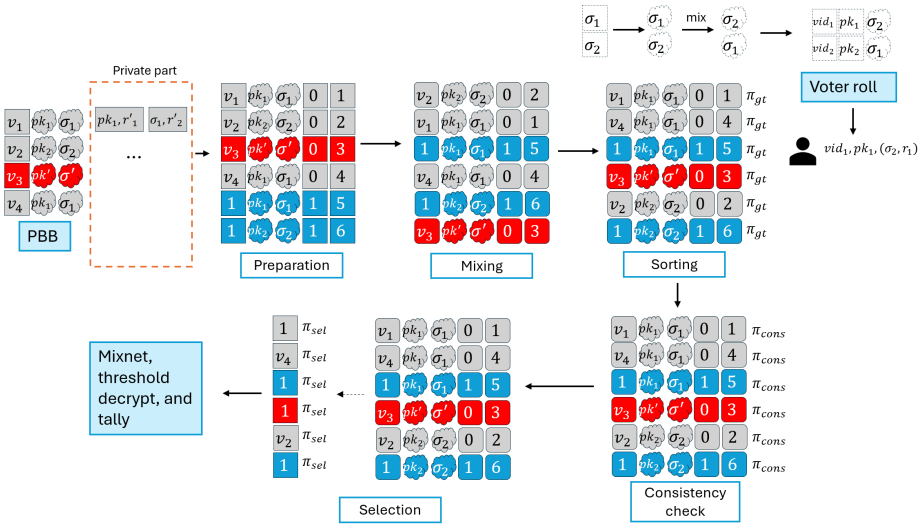

Fig. 7: FiltrumVote with everlasting privacy by anonymizing ballots. The cloud icon represents commitments.

C.2 B-ID

Another approach to bring everlasting privacy to voting systems is to protect the vote directly, adopting the Privacy-Preserving Audit Trail (PPAT) [9]. This approach essentially introduces a Secret Bulletin Board (SBB), available only to the EA and Trustees (SBB). Then, the PBB contains only the commitment to the vote $\langle v_i \rangle$, while SBB contains the encryption of the vote $[\![v_i]\!]_T$. In addition, to enable Trustees to open the commitment $\langle v_i \rangle$ at the end of the tally process, each ballot in SBB also includes an encryption of the commitment randomness $[\![r_i]\!]_T$. Tuples in PBB and SBB are processed in parallel. In the end, the Trustees decrypt $[\![v_i]\!]_T$ and $[\![r_i]\!]_T$ from the valid votes in the SBB and publishes

it. Independent auditors check if v_i and r_i indeed open the commitment $\langle v_i \rangle$ in PBB.

Although B-ID has stronger everlasting privacy protection than B-ANON, it requires publishing the opening of the vote $\langle v_i \rangle$ in the end. Thus, the EA must verify in the submission step if $[\![r_i]\!]_T$ really encrypts an opening of $\langle v_i \rangle$. Currently, this requires the use of a primitive called *CCEVA* (Commitment Consistent Encryption with Validity Augmentation). Unfortunately, to the best of our knowledge, the only available CCEVA are based on Pailler (which increases the parameters significantly) or in pairings (which increases complexity). Building an CCEVA based only on elliptic curve cryptography is an open problem.

C.3 Discussion

The most suitable approach for achieving everlasting privacy ultimately depends on the acceptable level of risk and implementation complexity. If the EA prefers to retain the underlying ElGamal cryptosystem, together with its associated libraries and existing infrastructure, the B-ANON approach may be preferable despite its higher risk of ballot identification and the additional information it may leak. In contrast, if the EA prioritizes a stronger notion of everlasting privacy, the B-ID approach may be more appropriate, despite the added complexity of incorporating a Paillier or pairing-based cryptosystem and maintaining two parallel bulletin boards, the PBB and the SBB.