VoteXX: Extreme Coercion Resistance

David Chaum
Richard T. Carback
Mario Yaksetig
xx network

Jeremy Clark
Mahdi Nejadgholi
Concordia University
Canada

Bart Preneel
COSIC, KU Leuven and imec
Belgium

Alan T. Sherman
Cyber Defense Lab, University of Maryland, Baltimore County (UMBC)
USA

Filip Zagorski
University of Wroclaw
Poland

Bingsheng Zhang
Zeyuan Yin
Zhejiang University
China

ABSTRACT

We solve a long-standing challenge to the integrity of votes cast without the supervision of a voting booth: "*improper influence*," which we define as any combination of vote buying and voter coercion. In comparison with previous proposals, our system is the first in the literature to protect against a strong adversary who learns all of the voter's keys—we call this property *"extreme coercion resistance."* Our approach allows each voter, or their trusted agents (which we call "*hedgehogs*"), to *"nullify"* (effectively cancel) their vote in a way that is unstoppable and irrevocable, and such that the nullification action is forever unattributable to that voter or their hedgehog(s). We demonstrate the security of VoteXX in the universal composability model. Additionally we provide concrete implementations of sub-protocols—including inalienable authentication, decentralized bulletin boards, and anonymous communication channels—that are usually left as abstract assumptions in the literature.

As in many other coercion-resistant systems, voters are authorized to vote with public-private keys. Each voter registers their public keys with the *Election Authority (EA)* in a way that convinces the EA that the voter has complete knowledge of their private keys. Voters concerned about losing their private keys can themselves, or by delegating to one or more hedgehog(s), monitor the bulletin board for malicious ballots cast with their keys, and can act to nullify these ballots in a privacy-preserving manner with zero-knowledge proofs.

In comparison with previous proposals, our system makes fewer assumptions and protects against a stronger adversary. For example, VoteXX makes none of the following assumptions made by previous systems: the voter must complete registration before being coerced; the election will not close before the voter can cast a ballot after coercion; the voter needs to generate a fake password to evade coercion; and the voter knows an honest Election Authority official.

CCS '23, November 26–30, 2023, Copenhagen, Denmark

CCS CONCEPTS

• **Applied computing → Voting / election technologies**; • **Theory of computation → Cryptographic protocols**; • **Security and privacy → Privacy-preserving protocols**; *Distributed systems security*; Web protocol security; *Software security engineering*.

KEYWORDS

Anonymous communication system, extreme coercion resistance, decentralized election authority, election security, hedgehog, high-integrity voting system, improper influence, Internet voting, mixnet, mix network, nullification, online voting, remote voting, voter-verifiable elections, VoteXX.

ACM Reference Format:
David Chaum, Richard T. Carback, Mario Yaksetig, Jeremy Clark, Mahdi Nejadgholi, Bart Preneel, Alan T. Sherman, Filip Zagorski, Bingsheng Zhang, and Zeyuan Yin. 2023. VoteXX: Extreme Coercion Resistance. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 21 pages.

1 INTRODUCTION

For over 150 years, the voting booth helped prevent voters from being bribed and coerced. For example, a controlling family member might coerce a voter by observing them vote, if votes are cast online from home or by mail. The booth, however, is becoming untenable as information technology provides the means for people to vote more frequently and conveniently without booths, including using combinations of mailed paper forms and online interactions. Moreover, growing use of technology facilitates vote buying and voter coercion with electronic payments, live video streaming from voter phones, and various types of online threats.

Three daunting challenges make Internet voting difficult: (1) The lack of a secure physical voting precinct facilitates improper influence, including vote selling and coercion. (2) Malware on a voter's device (e.g., phone) might undetectably modify votes and spy on the voter. (3) Determined adversaries might try to launch an online attack, including causing outages. Of these challenges, the most elusive has been mitigating improper influence.

We present a solution to the problem of *improper influence* in voting without booths that enables any voter to *"nullify"* (effectively cancel) their vote in a way that is unstoppable, irrevocable, and forever unattributable to that voter. Our approach allows each voter to recruit one or more trusted agents, which we call "*hedgehogs*." The voter, or their hedgehog(s), can nullify the vote by proving

Table 1: Properties of related work for resisting improper influence in online *end-to-end (E2E)* verifiable elections. Properties are fully present (●), partially present (◐), or not present (○). It is best to receive ● for each property. Decoy ballots act indirectly against influence (⊙).

Influencer: System resists coercion when the influencer: (0) acts before/during registration; (1) colludes with the EA; (2) colludes with hardware manufactures; (3) acts at any time; (4) learns all information stored by the voter, including all keys required by the protocol (*i.e.,* mitigates extreme coercion); (5) learns every action taken by the voter. **Other:** (6) voter can undo coercion undetectably; (7) system is inexpensive; (8) system has low cognitive burden; (9) system has security proof (none/game-based/UC).

		0	1	2	3	4	5	6	7	8	9
Type	**Example**	**Influencer**						**Other**			
Baseline (coercible)	Helios (2008) [2]	○	○	○	○	○	○	○	●	●	◐
Fake credentials	JCJ (2005) [27]	○	◐	●	○	○	○	●	●	●	◐
Masked ballots	WeBu09 (2009) [43]	○	●	●	○	○	○	○	●	○	●
Panic passwords	Selections (2011) [10]	○	●	●	○	○	○	●	●	◐	◐
Decoy ballots	RS-Voting (2012) [8]	●	◐	●	⊙	⊙	○	○	●	◐	●
Secure hardware	AOZZ (2015) [3]	●	◐	○	○	◐	○	●	○	●	●
Re-voting (E2E)	VoteAgain (2020) [32]	●	○	●	○	○	○	●	●	●	◐
Hedgehogs	VoteXX (2022)	●	●	●	●	●	○	○	●	●	●

knowledge of the voter's private key using a *zero-knowledge proof (ZKP)* without revealing the private key. This paper provides details for these ideas, which we introduced in 2022 [1].

Hedgehogs can be recruited before or during the election, from the voter's acquaintances or using a service selected on reputation. Hedgehogs can prove to the voter that they perform their services correctly. We call a "coercer" any entity who obtains a voter's key by coercion or bribery, whereas a "hedgehog" is an entity the voter trusts and to whom the voter voluntarily provides the key to protect the voter against coercers.

We accept that certain types of coercion are impossible to prevent in practice: a coercer can generally block a voter from registering for an election, and if a coercer posses all knowledge and attributes of the voter, they cannot be distinguished from the voter. Our approach differs from previous approaches with end-to-end verifiability (see Section 2)—e.g., revoting, fake credentials, and decoy ballots—by protecting against what we believe to be the strongest possible adversarial model that can be realistically protected against. Specifically we assume adversaries can learn all voter secrets and observe all voter interactions with the system, excluding interactions with the hedgehogs which distinguish the voter from the coercer. We call this protection "*extreme coercion resistance*."

An essential component of our system is ensuring voters actually know their private keys at registration time. Recent work by Kelkar *et al.* [28] argues that proofs of knowledge, signing challenge messages, and other techniques that often appear in voting systems do not rule out the possibility that private keys are encumbered by an adversary (*e.g.,* using hardware enclaves) so that voters can access enough of the key to satisfy the protocol without actually knowing it. Their alternative is "*complete knowledge*." Registration in our system contains a probabilistic test that the voter has complete knowledge of their secret key.

This paper presents an architecture, design, implementation, and *universal composability (UC)* [7] security proof (see Appendix C) of our voting system, called *VoteXX*. The main feature of VoteXX is that it protects against extreme coercion, which we formally define in terms of UC ideal functionality (see Section 3.2). Our VoteXX protocols include comprehensive mechanisms to handle all of the security requirements, including, for example, inalienable authentication, which many other voting systems simply assume without providing constructions. We describe the user experience for several settings, which experiences are intuitive and require few steps. We have implemented the entire VoteXX system and made all of our sourcecode publicly available as an artifact (see Section 7). Performance analysis and benchmarking show that the system is highly practical (see also Section 8.4).

Our primary contributions are: (1) We introduce the new notions of nullification and hedgehogs, and present a new solution to improper influence based on them. (2) We give cryptographic protocols realizing nullification, and show how nullification can be applied to several voting settings, including vote-by-mail and online. (3) We present a new fully-decentralized scalable voting system, *VoteXX*, including registration, voting, nullification, and tallying. (4) We describe our implementation of VoteXX, which uses an *anonymous communication system (ACS)* for registration, vote casting, and other communication. (5) We provide a formal statement and UC proof of VoteXX's ballot secrecy, coercion resistance, and tally integrity. In addition, while other systems complicate registration and vote casting, our approach allows simple registration and vote casting by keeping nullification separate. Consequently, our system can be used as an overlay in conjunction with other approaches, such as re-voting and decoy ballots.

In the rest of this paper, we compare our approach with those of previous work, detail our adversarial model, give our problem specification, show the VoteXX architecture, define the VoteXX cryptographic protocols, describe voter interfaces for several settings including vote-by-mail and online, mention possible extensions to VoteXX, sketch the VoteXX implementation and discuss its performance, and explain the significance of our work. Appendix C gives our UC proofs.

Throughout, we use the terms "coercion" and "improper influence" synonymously.

2 COMPARISON TO PREVIOUS WORK

Coercion resistance guarantees that each voter may vote freely. Informally, a voting system is *coercion resistant* if and only if no ballot is "*counted as coerced*," that is, no voter can prove to any coercer that the voter cast a counted ballot according to the coercer's instructions. As explained in Sections 3.2 and C.2, we uniquely adopt a very strong form of coercion resistance, which we call *extreme coercion*, in which the coercer learns all of the voter's keys. By contrast, other researchers assume only weaker forms of coercion, such as *semi-honest coercion* (receipt-freeness) in which the the voter must follow the voting protocol (see Table 1), or *active coercion* in which the voter can interact with the coercer during the voting protocol (see Table 1). Some researchers aim only to detect coercion rather than to mitigate it (e.g., Caveat Coercitor [20]).

In an unpublished manuscript, Smyth [39] surveys four definitions of coercion resistance and finds that "coercion resistance has not been adequately formalized." According to Smyth, three of the definitions are too weak, and the general definition by Küsters [30] is complex and too strong. His observations are controversial but demonstrate that settling definitions is still an elusive goal. Similarly, there remains some debate on the definition of receipt freeness [15]

Previous work often makes strong assumptions: the voter knows an honest *Election Authority (EA)* official [11]; the voter needs a special device to evade coercion [4, 5, 11, 27]; the voter needs to perform mental arithmetic to evade coercion [43]; the voter needs to generate a fake password to evade coercion [10, 16]; the voter must complete registration before being coerced [27]; the election will not close before the voter can cast a ballot after coercion [32, 40, 42]; and the probability of successful coercion is lowered by flooding voters with decoy ballots [8]. VoteXX makes none of these assumptions.

We do assume the voter can use an *untappable channel*, as all coercion-resistant systems must—if an adversary can always influence the voter, they are indistinguishable from the voter [24]. Some systems establish windows for this channel, such as during registration, or after coercion occurs. VoteXX is as flexible as it could be. The channel is used once or twice between the voter and each hedgehog (who can be any person in the world): first to induct the hedgehog (any time before the end of the election), and possibly second to signal the hedgehog (after coercion and before the end of the election).

VoteXX guarantees that the voter is able to nullify their coerced vote. Unlike some systems, in VoteXX, the voter cannot change their coerced ballot selection. VoteXX can be used as an overlay, providing an additional coercion-resistant mechanism to others already in place. Thus, VoteXX can support re-voting (as outlined in our protocol description): if a voter were unable to re-vote (due to coercion at the end of the election), nullification would be a failsafe. Similarly, VoteXX can be used together with decoy ballots.

Table 1 compares our solution to previous proposed mechanisms. We do so by scoring each mechanism with regard to five properties of the influencer and five other properties. We state each assumption and property positively, meaning it is better to receive ● than ○. Appendix B explains our assumptions and the basis for our scoring.

In any system, an adversary could always prevent a voter from voting. In this sense, VoteXX achieves an optimal solution. Furthermore, we conjecture, that for Column 6 in Table 1, no system that resists extreme coercion can also undo coercion undetectably.

3 SYSTEM OVERVIEW

In VoteXX, each voter has a public-private key pair for "YES" votes, and another such pair for "NO" votes. Without revealing their private keys, each voter registers their public keys with the EA. Each voter may share their keys with one or more hedgehogs. During nullification, the voter, or one or more of their hedgehog(s), can interact with the ACS to nullify a vote by proving knowledge of one of the voter's private keys via a ZKP. We describe a fully decentralizable implementation of VoteXX, including its public *bulletin board (BB)*, which could be implemented on a blockchain.

3.1 Adversarial Model

The adversary could be anyone—including a voter or an EA trustee, located close to or far away from their target. The adversary might be covert or overt. The adversary's goal might include any or all of the following: tamper with the tally, influence a voter's ballot choice through coercion, learn how a voter voted, or disrupt or discredit an election. The adversary can engage in coercion at any time, including before or during voter registration.

We assume a secure ACS that protects against traffic analysis. Examples include TOR with hidden services [41], I2P [25], xx network [45], and Oxen [35]. We further assume that the adversary cannot defeat standard cryptographic functions and protocols, including encryption, digital signatures, cryptographic hashing, pseudorandom number generation, and ZKPs. We assume an untappable channel between the voter and their hedgehog(s), as explained in Section 2

3.2 Ideal Functionality

A foundational component of our UC proof (Appendix C) is the *voting ideal functionality* $\mathcal{F}_{\text{Vote}}^{n,k,t}$, which we now introduce and define in Fig. 1.

The ideal functionality. The voting ideal functionality $\mathcal{F}_{\text{Vote}}^{n,k,t}$ has four phases: preparation, registration, voting, and tally. In the voting phase, $\mathcal{F}_{\text{Vote}}^{n,k,t}$ receives ballots from the voters and records them. In particular, $\mathcal{F}_{\text{Vote}}^{n,k,t}$ accepts a special type of request: "nullify." Upon receiving a nullify request, the former choice of the voter will not be counted in the final tally.

Extreme coercion. In our UC model, the adversary has the power of extreme coercion. When the adversary $\mathcal{A}$ sends an "extreme coercion" request to a voter, V_i, V_i will hand his state to $\mathcal{A}$ and follow $\mathcal{A}$'s instructions, but V_i can still communicate with his hedgehog(s) H_i secretly.

Connection with the properties. It is easy to see that our UC definition implies the basic properties of a secure voting scheme. First, $\mathcal{F}_{\text{Vote}}^{n,k,t}$ does not leak the ballot of a voter to anyone else, so it implies ballot privacy. Second, as mentioned above, the ideal deception is able to nullify the ballot and the coercer cannot know if the

Functionality $\mathcal{F}_{\text{Vote}}^{n,k,t}$

The functionality $\mathcal{F}_{\text{Vote}}^{n,k,t}$ interacts with a set of voters $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, a set of hedgehogs $\mathcal{H} := \{\mathsf{H}_1, \ldots, \mathsf{H}_n\}$, a set of trustees $\mathcal{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$, the *Election Authority (EA)*, and the adversary $\mathcal{S}$. Internally it keeps variables status, ballots, τ, and $\mathcal{J}$. Let $\mathcal{P}_{\text{cor}}$ be the set of corrupted parties.
Initially, set status $:= 0$, ballots $:= \tau := \mathcal{J} := \emptyset$.

Preparation:

- Upon receiving (Start, sid) from the trustee $\mathsf{T}_j \in \mathcal{T}$, set $\mathcal{J} := \mathcal{J} \cup \{\mathsf{T}_j\}$, and send a notification (Start, sid, T_j) to the adversary $\mathcal{S}$. (If status $\neq 0$, then ignore the request.)
- Upon receiving (Begin, sid) from the EA, if $|\mathcal{J}| < k$ ignore the request. Otherwise, send a notification (Begin, sid) to the adversary $\mathcal{S}$, and set status $:= 1$. (If status $\neq 0$, then ignore the request.)

Registration:

- Upon receiving (Register, sid) from the voter V_i, send (Register, sid, V_i) to the adversary $\mathcal{S}$. (If status $\neq 1$, then ignore the request.)
- Upon receiving (EndReg, sid) from EA, send (EndReg, sid) to the adversary $\mathcal{S}$ and set status $:= 2$. (If status $\neq 1$, then ignore the request.)

Voting:

- Upon receiving (Vote, sid, x) from a voter $\mathsf{V}_i \in \mathcal{V}$, set ballots$[i] := x$ (x=YES/NO), and send (VoteNotify, sid, V_i) to the adversary $\mathcal{S}$. (If status $\neq 2$, then ignore the request.)
- Upon receiving (EndVote, sid) from EA, compute $\delta \leftarrow$ TallyAlg(ballots) (Cf Fig. 6). Send (PreTally, sid, δ) to the adversary $\mathcal{S}$. Set status $:= 3$. (If status $\neq 2$, then ignore the request.)
- Upon receiving (Nullify, sid) from a voter $\mathsf{V}_i \in \mathcal{V}$ or V_i's hedgehog H_i, set ballots$[i] :=$ nullify. Send (NullifyNotify, sid) to the adversary $\mathcal{S}$. (If status $\neq 3$, then ignore the request.)

Tally:

- Upon receiving (Tally, sid) from EA, compute $\tau \leftarrow$ TallyAlg(ballots) (Cf Fig. 6). Send (Tally, sid, τ) to the adversary $\mathcal{S}$. (If status $\neq 3$, then ignore the request.)
- Upon receiving (Result, sid) from any party P, if $\tau := \emptyset$, then ignore the request, otherwise return (Result, sid, τ) to the requester.

Figure 1: Functionality $\mathcal{F}_{\text{Vote}}^{n,k,t}$.

coercion was successful, so our definition implies coercion resistance. Third, $\mathcal{F}_{\text{Vote}}^{n,k,t}$ ensures that the tally procedure is performed correctly, so it implies verifiability.

3.3 Problem Specification

Our main requirement is a coercion-resistant remote voting system that achieves a level of security at least as strong as that for a precinct-based in-person voter-verifiable paper secret-ballot system. The system must maximize the ability to prevent or remediate serious failures by eliminating undetectable attacks, preventing scalable "wholesale" attacks, and making "retail" attacks as difficult as possible. The key requirements, specific to our context, are *coercion resistance*, *malware resistance*, and *availability*.

Coercion resistance. An adversary cannot be convinced that the voter's ballot is "counted as coerced," that is, counted the way the coercer instructed the voter to vote. This property is related to *ballot secrecy* but we assume that the adversary can watch the voter vote or vote for them. The adversary, however, cannot be sure how that vote is counted, so they have no incentive to threaten or pay the voter to vote a certain way. While rarely a significant issue in polling place elections, this problem is much more important in uncontrolled environments such as absentee voting or Internet voting.

Malware resistance. Any modification of the hardware or software that changes the result must be detectable. This property is similar to *software independence* but with the caveat that a version of the software exists without the undetected change before or after the election. In other words, the adversary does not, for all time, control everything read or written to all devices used by the voter for voting.

Availability. The system must not have single points of failure. It should resist denial of service attacks, and no single entity should be able to prevent completion of the election.

3.4 System Architecture

As shown in Fig. 2, we describe VoteXX in terms of the following entities and elements. There are n *voters* $v_1, v_2, \ldots, v_n$ who interact with a publicly readable *BB*, which is a distributed ledger such as a blockchain. The writing interactions take place via an ACS. The ACS disassociates the device, physical location, and other associated metadata by all clients posting to the BB, protecting the metadata of voters and hedgehogs, as well as sensitive election authority equipment. Read operations can take place through the ACS or via a direct interaction with the BB. Each voter may have one or more trusted *hedgehog(s)*. Each hedgehog interacts with the BB via the ACS. The EA comprises a set of independent and non-colluding (up to a threshold) entities called *trustees*. The trustees of the EA are authoritative over registration, voting, and tallying. The EA can read and write to the BB via the ACS. The system includes a set of *auditors* who can read from the BB and verify the correctness of operations performed by the EA and via the ACS.

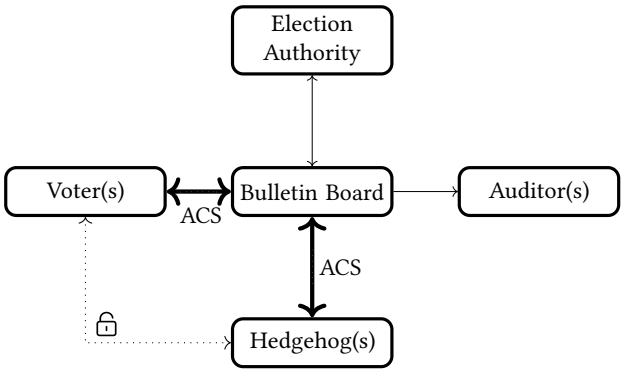

Figure 2: VoteXX Architecture. Arrows represent information flows (i.e., read/write) between system entities. Thick lines represent communications that take place over the ACS. The EA can write to the BB directly or via the ACS. Voters submit (encrypted) ballots over the ACS, read from the BB, and share their secret key over an untappable channel (dotted) to their hedgehog(s). The hedgehogs submit a ZKP to nullify the corresponding vote(s) over the ACS. Auditors read from the BB.

4 PROTOCOLS

Protocol Boxes 1–3 explain the four main stages of the VoteXX protocols: registration, voting, and tallying (including nullification). Section 5 explains nullification in more detail.

The VoteXX protocol assumes a number of cryptographic primitives that are common in the voting literature. All operations are performed in the same elliptic curve group, where the decisional *Diffie-Hellman (DDH)* problem (and by extension, the discrete logarithm problem) is hard. Digital signatures are performed with the Schnorr signature scheme. Encryption is performed with ElGamal [14], which can be augmented with *distributed key generation (DKG)* and threshold decryption (for m out of n key holders [37]).

We use standard Σ−Protocols to prove knowledge of discrete logarithms (Schnorr [38]), knowledge of representations (Okamoto [34]), and knowledge of Diffie-Hellman tuples (Chaum-Pedersen [9]), which also corresponds to ElGamal re-randomizations and decryptions. We also use techniques to allow the trustees to compute jointly, verifiably (*i.e.,* produce Σ−Protocol proofs), and privately on ElGamal ciphertexts the following: (i) a random shuffle of ciphertexts (Verificatum), and (ii) the evaluation of an *exclusive-or (XOR)* operation based on its logic lookup table (mix and match [26]).

Protocol 1 describes registration. Registration can be re-opened by re-running set-up. Once a voter key is registered, it can be used in later registration periods. Protocol 1 is one way of performing registration, but any method that results in a posting of the voter's public key (in encrypted format) is fine. A simple way is for the voter to have the private key (full entropy, not based on a passphrase) on a hardware device and provide the public key. One problem that we tackle in Protocol 1 is providing assurance that the voter actually knows their private key—and it is not, for example, supplied by a coercer. This assurance is one of two properties of so-called *inalienable authentication*. The other property is that the adversary cannot impersonate the voter. Other authors do not provide concrete constructions for inalienable authentication; some simply tacitly assume it in their proofs of coercion resistance. In VoteXX, we provide a concrete instance of the first half of an unalienable authentication protocol, and we present a voting protocol that does not need the second half. That is, we care only that the voter knows their secret key—if the adversary also knows it too, we can still achieve coercion resistance.

Voting performs a straightforward signature using a registered key (see Protocol 2). At the end of registration, voter keys are unlinked from their identity. Until the election closes, votes are encrypted to preserve the secrecy of the tally, and ballots are submitted through the *ACS* to unlink them from the voter communication metadata.

5 NULLIFICATION

We explain the nullification protocol in detail. First, we present an overview. Second, we give the construction of the nullification ZKP and propose a novel succinct ZKP with $O(\log n)$ proof size, where n is the number of total ballots.

5.1 Overview

The tallying process (Protocol 3) includes our novel nullification technique. Consider a list of public keys that voted YES and assume the hedgehog wants to nullify one of them. It cannot point out which key it wants to nullify or the coercer would know the voter is working with (or is personally acting as) a hedgehog to intervene. So the hedgehog must hide its flag ($[\![1]\!]$) in a set of false flags ($[\![0]\!]$) for each YES key in the tally. We could allow the hedgehog to choose a fixed-sized subset of β keys at random to serve as an anonymity set, which improves performance but sacrifices full anonymity (*cf.* [10]). For simplicity, the protocol boxes do not explain that, for nullification, we use exponential ElGamal [14] instead of standard ElGamal used in registration and voting (under the same election master key).

If a hedgehog flags a key with ($[\![1]\!]$), it must know the associated private key; otherwise, any hedgehog could nullify any vote. However, if it submits a false flag ($[\![0]\!]$), it does not need to know the associated key. Anyone can serve as a dummy hedgehog by submitting a full set of false flags. To enforce these constraints, the hedgehog must construct a ZK proof to prove that: [for each flag, (it is an encryption of 0) or (it is an encryption of 1 and I know $\mathsf{sk}_{\mathsf{no}}$ corresponding to this $\mathsf{pk}_{\mathsf{no}}$)]. We will describe the NIZK proof for the above statement in Section 5.2.

Once a hedgehog computes and submits a set of flags (along with the NIZK proof Π), Protocol 3 simplifies the description by having the EA wait to perform Steps 1–2 after the nullification period. In practice, it should not wait—the process is quadratic work (number of hedgehogs times number of voters) and subject to "board-flooding" attacks [29]. The EA must process the nullifications as they arrive; that is, use "concurrent authorization" [16]. Doing so is possible. When a new set of flags arrives, the EA checks each proof and computes the XOR between the submitted flag. The EA also computes the accumulation of previous flags—each of these two steps is parallelizable for each flag. Thus, when nullification closes, the only remaining task is to threshold decrypt the accumulation of flags, which process is linear in the number of votes.

5.2 The Nullification ZKP

We provide a formal description of the nullification ZK proof. It is well known that Σ−Protocols can be *stacked through conjunction and disjunction (CDS)* [13, 18]. We first present the CDS-composition ZKP and then propose a novel succinct ZKP.

The CDS-composition ZKP. The CDS-composition ZKP takes a voter's public key pk and makes a disjunctive proof that either Case 1 `OR` Case 2 is true: In Case 1, the hedgehog proves (flag = $[\![0]\!]$). For exponential ElGamal, assume $\langle c_1, c_2\rangle = \mathsf{Enc}(m) = \langle g^r, g^m y^r\rangle$ for generator g, public key y, and message m. A proof it encrypts $\hat{m}$ is equivalent to proving $\left\langle g, c_1, y, c_2\hat{m}^{-1}\right\rangle$ is a DDH tuple, which can be done with the Chaum-Pedersen Σ−Protocol. Call this subproof A. In Σ−Protocol format, its transcript is $\langle a_A, e_A, z_A\rangle$.

In Case 2, the hedgehog proves a conjunctive statement: (flag = $[\![1]\!]$) and it knows sk, which corresponds to pk for the associated voter's public key. Call the subproof that (flag = $[\![1]\!]$) B. It is implemented the same as in subproof A, with transcript $\langle a_B, e_B, z_B\rangle$. Call the proof of knowledge of sk subproof C, which can be implemented with a Σ−Protocol due to Schnorr: $\langle a_C, e_C, z_C\rangle$. To summarize, the hedgehog proves: $\Pi :=$ [A `OR` (B `AND` C)] for each flag.

Further, the resulting proof can be made non-interactive (typically in the random oracle model with the Fiat-Shamir heuristic [17],

Registration is an in-person ceremony between the voter, using a *voting client* device, and an officer for the EA. At completion, the voter registers two public keys $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}} \rangle$, which are not learned by the EA officer and will be used to vote YES and NO, respectively. The keys are for a digital signature. They are based on a passphrase that can be regenerated from any voting client. The EA additionally does not learn the passphrase but has high assurance through the protocol that the human voter knows the passphrase.

Registration Set-up.

Registration uses a trapdoor commitment scheme. The commitment aspect allows the voter to present her passphrase in a hidden form to the EA and answer queries about specific characters within it. The trapdoor is revealed after registration closes and allows each voter to convert the format of their commitments into the format of a public key.

(1) The generator g_0 is a parameter of the election.
(2) The EA computes a generator g_1 as follows: each trustee $T, T', T'', \ldots$ privately chooses one random value a_1, reveals $g_0^{a_1}$, and proves knowledge of a_1 with a Schnorr Σ−Protocol. Then $g_1 = g_0^{(a_1 + a_1' + a_1'' + \ldots)}$.
(3) This process is repeated, with new random a_i values, to complete a set of N generators: base $\leftarrow \langle g_0, g_1, g_2, \ldots, g_{N-1} \rangle$. The same base is used for all voters in a registration period.
(4) Call the set of all a values (split across the trustees): trapdoor.

Registration.

(1) Each voter generates two N-character passphrases (for YES and NO). Steps 2–4 describe the process for the first passphrase and are repeated for the second.
(2) The voting client parses the passphrase as a sequence of Base64 characters $\langle c_0, c_1, c_2, \ldots, c_N \rangle$ and computes its deterministic commitment using base: $\mathsf{passCommit} \leftarrow \langle g_0^{c_0} \cdot g_1^{c_1} \cdot g_2^{c_2} \cdot \ldots \cdot g_{N-1}^{c_{N-1}} \rangle$.
(3) The voting client sends $[\![\mathsf{passCommit}]\!]$ to the EA, which is an encryption of passCommit under the EA's threshold encryption scheme.
(4) The EA officer issues a challenge like: "Reveal Character 4." The voter responds "F." The EA client computes $\mathsf{disclosedChar} \leftarrow ([\![\mathsf{passCommit}]\!]/g_4^{\mathsf{F}})$. The voting client proves knowledge of a representation of disclosedChar using a Σ−Protocol. This step is repeated to build confidence that the voter knows the passphrase, but bounded in repetitions to protect the passphrase.
(5) The EA client posts $\langle \mathsf{VoterID}, [\![\mathsf{passCommit}_{\mathsf{yes}}]\!], [\![\mathsf{passCommit}_{\mathsf{no}}]\!] \rangle$ to the BB.

Registration Finalization.

(1) After the registration period, the EA takes the list of $\langle \mathsf{VoterID}, [\![\mathsf{passCommit}_{\mathsf{yes}}]\!], [\![\mathsf{passCommit}_{\mathsf{no}}]\!] \rangle$ entries, removes the VoterID component, and verifiably shuffles, threshold-decrypts, and posts $\langle \mathsf{passCommit}_{\mathsf{yes}}, \mathsf{passCommit}_{\mathsf{no}} \rangle$ for each (now anonymous) voter.
(2) Each trustee $T, T', T'', \ldots$ reveals their values producing trapdoor.
(3) Each voter uses trapdoor to reformat their two passCommit values into key pairs $\langle \mathsf{sk}, \mathsf{pk} \rangle$ such that $\mathsf{pk} = \mathsf{passCommit} = g_0^{\mathsf{sk}}$ as follows. Consider generator g_i and let $\alpha_i = a_i + a_i' + \ldots$. With this notation, $\mathsf{sk} = c_0 + \alpha_1 \cdot c_1 + \alpha_2 \cdot c_2 \ldots$.
(4) Given that $\langle \mathsf{passCommit}_{\mathsf{yes}}, \mathsf{passCommit}_{\mathsf{no}} \rangle = \langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}} \rangle$, the EA holds an anonymized list, which we call the Roster, of $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}} \rangle$ keys for each registered voter.

Protocol 1: Registration Protocol.

Voting.

Each voter completes voting online. At completion, each voter will have submitted their ballot using a passphrase from registration.

(1) The value nonce is a parameter of the election.
(2) To mark a ballot for YES, the voter uses their YES passphrase to generate $\mathsf{sk}_{\mathsf{yes}}$ and uses this key to sign n_0: $\sigma_{\mathsf{yes}} \leftarrow \mathsf{Sign}(\mathsf{nonce})$. Corresponding values are used to vote NO.
(3) The voter uses the EA's threshold encryption scheme to compute ballot $\leftarrow \langle [\![pk_{\mathsf{yes}}]\!], [\![\sigma_{\mathsf{yes}}]\!], \pi_{\mathsf{ppk}} \rangle$, where each group element of σ is individually encrypted and π_{ppk} is a proof of plaintext knowledge using the Chaum-Pedersen Σ−Protocol.
(4) The voter submits ballot over the ACS to the BB. The EA marks it as invalid if it is an exact duplicate or if the proofs are invalid.

Protocol 2: Voting Protocol.

in its strong form [6], but other heuristics exist [23]). Specifically, the prover generates a single challenge $\hat{e}$ for Π. To handle the conjunction within Case 2, $e_B = e_C$; for the disjunction across the cases, $\hat{e} = e_A + e_B$. In Case 1, the prover computes $\langle a_A, e_A, z_A \rangle$ and simulates $\langle a_B, e_B, z_B \rangle$ and $\langle a_C, e_B, z_C \rangle$. In Case 2, the prover simulates $\langle a_A, e_A, z_A \rangle$ and computes $\langle a_B, e_B, z_B \rangle$ and $\langle a_C, e_B, z_C \rangle$.

A new succinct ZKP. Here, we propose a novel succinct nullification ZKP with $O(\log N)$ proof size, where N is the number of total ballots. Assuming that in the nullification phase, each nullification request nullifies only one ballot. To nullify a ballot, the hedgehog will form a list of encrypted "flags," where there is one encryption of 1 and the other flags are encryptions of 0. The hedgehog needs to prove in ZK that (i) there is one encrypted flag containing $[\![1]\!]$ and the others are $[\![0]\!]$, and (ii) I know the corresponding sk.

Formally, let h denote the ElGamal public key, and let $[\![x; r]\!]$ denote exponential ElGamal encryption with explicit randomness, i.e., $[\![x; r]\!] := (g^r, g^x h^r)$. Let ck denote the Pedersen commitment key, and let Com denote Pedersen commitment, i.e., $\mathsf{Com}(x; r) := g^x \mathsf{ck}^r$. Denote the public keys in yesVotes as $\mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}$. Denote the encrypted flags as $E_0, \ldots, E_{N-1}$. Let $n := \lceil \log N \rceil$. We will give a ZK protocol for the relation

Provisional Tally.

After the voting period ends, the EA produces a verifiable provisional tally.

(1) The EA takes the list of $\langle [\![\mathsf{pk}]\!], [\![\sigma]\!] \rangle$, then threshold-decrypts them: $\langle \mathsf{pk}, \sigma \rangle$.
(2) For each ballot, the ballot is marked invalid if σ does not verify under its corresponding pk.
(3) For each valid signature, pk is matched to its entry on the Roster. The EA determines if it is a YES or NO key, and counts the vote only if it is the only ballot cast that corresponds to that roster entry. (Since ballots are not shuffled, other policies are feasible such as counting the most recent vote.)

Nullification.

The goal of nullification is to allow voters to modify their cast ballots, particularly in the case of coercion. Unlike other protocols, voters can enlist the help of others parties, called hedgehogs. The nullification period runs after the provisional tallying. If the provisional tally contains $\mathsf{pk}_{\mathsf{no}}$, it can be nullified using $\mathsf{sk}_{\mathsf{yes}}$ (the "opposite" key). In other words, casting a YES and nullifying a NO vote use the *same* key, as these two actions are aligned in their intention.

(1) At any convenient time, before or after voting, the voter covertly communicates with a hedgehog to develop a coercion-resistant strategy. For example, assume the following strategy: the voter wants to vote YES and reveals $\mathsf{sk}_{\mathsf{yes}}$ to the hedgehog, along with $\left\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}} \right\rangle$. They request the hedgehog engage in nullification if $\mathsf{pk}_{\mathsf{no}}$ is in the provisional tally.
(2) Using the Roster and set of valid signatures from the provisional tally, the EA reformats the election data into two lists. The first list establishes, in arbitrary order, the set of $\mathsf{pk}_{\mathsf{no}}$ keys from voters who cast valid votes for YES (call it yesVotes). The second list contains $\mathsf{pk}_{\mathsf{yes}}$ from voters who voted NO.
(3) For example, assume YES received six votes in the provisional tally. yesVotes consists of six $\mathsf{pk}_{\mathsf{no}}$ keys. If the hedgehog wants to nullify the fourth key, it prepares a list of encrypted "flags" marking the ballot it wants to nullify: $\langle [\![0]\!], [\![0]\!], [\![0]\!], [\![1]\!], [\![0]\!], [\![0]\!] \rangle$.
(4) The first encrypted flag corresponds to the first $\mathsf{pk}_{\mathsf{no}}$ in yesVotes. The hedgehog adds a proof to this list using the nullification ZK protocol. Concisely, the proof statement is: [for each flag, (it is an encryption of 0) or (it is an encryption of 1 and I know $\mathsf{sk}_{\mathsf{no}}$ corresponding to this $\mathsf{pk}_{\mathsf{no}}$)].

Final Tally.

After the nullification period ends, the EA produces a verifiable final tally.

(1) The EA takes all the encrypted flags for the first $\mathsf{pk}_{\mathsf{no}}$ key in yesVotes and computes its or under encryption using the mix and match SFE protocol [26]. It repeats this process for the remaining $\mathsf{pk}_{\mathsf{no}}$ keys.
(2) The EA takes the list of encrypted ored flags, sums them under encryption, and verifiabily threshold-decrypts the result. The EA subtracts this value from the number of YES votes in the provisional tally to produce the final tally for YES votes.
(3) The EA repeats Steps 1–2 for each $\mathsf{pk}_{\mathsf{yes}}$ key in noVotes.

Protocol 3: Tallying Protocol (including nullification).

$$\mathcal{R} = \{((\mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}, E_0, \ldots, E_{N-1}), r_0, \ldots, r_{N-1}, \ell, \mathsf{sk}) \mid \ell \in \{0, \ldots, N-1\} \ \wedge \ \mathsf{pk}_\ell = g^{\mathsf{sk}} \ \wedge \ E_\ell = [\![1; r_\ell]\!] \ \wedge \ E_i = [\![0; r_i]\!], i \neq \ell\}. \tag{1}$$

Following the idea of [21, 47], the prover first commits bit-wise to the binary representation of ℓ. The key observation is that there exists a data-oblivious algorithm that takes as input the binary representation of ℓ and generates a unit vector where the ℓth element is 1.

Concretely, the protocol can be split into two parts: The first part proves that there is one encrypted flag containing $[\![1]\!]$ and the others are $[\![0]\!]$, which is actually a unit vector proof [47]. The second part proves that the prover knows the corresponding sk, which can be proven by modifying the one-out-of-many proof [21]. The modification works as follows. The prover first computes the commitment of sk, denoted as c. Then, the verifier can compute $c_i := \mathsf{pk}_i / c$. Now $\{c_i\}$ is a vector satisfying $c_\ell = \mathrm{Com}(0)$ so that the one-out-of-many proof [21] can be applied. The prover needs to additionally prove that he knows the opening of c.

We specify the polynomial $p_i(x)$ used in the protocol. Following [21], we write $i = i_1 \ldots i_n$ and $\ell = \ell_1 \ldots \ell_n$ in binary, and we let δ_{ij} be Kronecker's delta, i.e., $\delta_{\ell\ell} = 1$ and $\delta_{i\ell} = 0$ for $i \neq \ell$. We let $f_j = \ell_j x + a_j$, let $f_{j,1} = f_j = \ell_j x + a_j = \delta_{1\ell_j} x + a_j$ and $f_{j,0} = x - f_j = (1 - \ell_j)x - a_j = \delta_{0\ell_j} x - a_j$. Then, $p_i(x) = \prod_{j=1}^n f_{j,i_j}$ has the form:

$$p_i(x) = \prod_{j=1}^{n} (\delta_{i_j \ell_j} x) + \sum_{k=0}^{n-1} p_{i,k} x^k = \delta_{i\ell} x^n + \sum_{k=0}^{n-1} p_{i,k} x^k. \tag{2}$$

Fig. 3 shows the ZK protocol for relation $\mathcal{R}$. By the Fiat-Shamir heuristic [17], it can be transformed into a NIZK proof.

Table 2 shows the prover computation, verifier computation, and proof size of our new succinct ZKP, compared with the CDS-composition ZKP.

Table 2: Prover computation, verifier computation, and proof size of our new succinct ZKP and CDS-composition ZKP.

	CDS	New succinct ZKP
Prover comp.	$5N$ Expos	$(N\lceil \log_2 N \rceil + N + 7\lceil \log_2 N \rceil + 7)$ Expos
Verifier comp.	$11N$ Expos	$(5N + 7\lceil \log_2 N \rceil + 6)$ Expos
Proof size	$5N$ $\mathbb{G}$ + $4N$ $\mathbb{Z}_q$	$(5\lceil \log_2 N \rceil + 2)$ $\mathbb{G}$ + $(3\lceil \log_2 N \rceil + 4)$ $\mathbb{Z}_q$

Theorem 5.1. *Assume that the DDH problem is hard. The protocol in Fig. 3 for relation $\mathcal{R}$ is a 4-move public coin ZK protocol with completeness, soundness, and special honest verifier ZK.*

Proof. For *completeness*, it is easy to see that $c_{\ell_j}^x c_{a_j} = \mathrm{Com}(f_j; z_{a_j})$ and $c_{\ell_j}^{x - f_j} c_{b_j} = \mathrm{Com}(0; z_{b_j})$ hold for $j \in \{1, \ldots, n\}$ and $g^{v_1} h^{v_2} = m \cdot c^x$ holds. Then, observe that $\prod_{j=1}^n f_{j,i_j}$ is a polynomial in the challenge x of the form $p_i(x) = \delta_{i\ell} x^n + \sum_{k=0}^{n-1} p_{i,k} x^k$. By the additive homomorphism of Pedersen commitment, $\prod_{i=0}^{N-1} c_i^{\prod_{j=1}^n f_{j,i_j}} \cdot \prod_{k=0}^{n-1} c_{d_k}^{-x^k} = \mathrm{Com}(0; z_d)$ always holds since c_ℓ is a commitment to 0. Similarly, denote $E_i = [\![e_i; r_i]\!]$, we have $\prod_{i=0}^{N-1} \left((E_i)^{x^n} \cdot [\![- \prod_{j=1}^n f_{j,i_j}; 0]\!] \right)^{y^i} \cdot \prod_{k=0}^{n-1} (D_k)^{x^k} = [\![\sum_{i=0}^{N-1} (e_i \cdot x^n - p_i(x) + \sum_{k=0}^{n-1} p_{i,k} x^k) \cdot y^i; R]\!] = [\![0; R]\!]$. Thus, the protocol is perfectly complete.

ZK protocol for relation $\mathcal{R}$

CRS: the ElGamal public key h, the Pedersen commitment key ck;
Statement: $\mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}, E_0, \ldots, E_{N-1}$;
Witness: $r_0, \ldots, r_{N-1}, \ell, \mathsf{sk}$ such that $\ell \in \{0, \ldots, N-1\} \wedge \mathsf{pk}_\ell = g^{\mathsf{sk}} \wedge E_\ell = [\![1; r_\ell]\!] \wedge E_i = [\![0; r_i]\!], i \neq \ell$.

Verifier:
- $V \to P$: Random $y \leftarrow \mathbb{Z}_q$.

Prover:
- Randomly pick $t \leftarrow \mathbb{Z}_q$, compute $c := \mathsf{Com}(\mathsf{sk}; t)$;
- For $i = 0, \ldots, N-1$, compute $c_i := \mathsf{pk}_i / c$.
- For $j = 1, \ldots, n$
  - Randomly pick $\tau_j, a_j, s_j, t_j, \rho_k \leftarrow \mathbb{Z}_q$;
  - Compute $c_{\ell_j} := \mathsf{Com}(\ell_j; \tau_j)$; $c_{a_j} := \mathsf{Com}(a_j; s_j)$; $c_{b_j} := \mathsf{Com}(\ell_j a_j; t_j)$;
  - Compute $c_{d_k} := \prod_{i=0}^{N-1} c_i^{p_{i,k}} \mathsf{Com}(0; \rho_k)$ (using $k = j-1$ and $p_{i,k}$ from Eq. 2);
  - Pick random $R_k \leftarrow \mathbb{Z}_q$ and compute $D_k := [\![\sum_{i=0}^{N-1} (p_{i,k} \cdot y^i); R_k]\!]$ (using $k = j-1$ and $p_{i,k}$ from Eq. 2);
- Randomly pick $s', t' \leftarrow \mathbb{Z}_q$, compute $m := \mathsf{Com}(s', t')$;
- $P \to V$: $(c, c_{\ell_1}, c_{a_1}, c_{b_1}, c_{d_0}, D_0, \ldots, c_{\ell_n}, c_{a_n}, c_{b_n}, c_{d_{n-1}}, D_{n-1}, m)$.

Verifier:
- $V \to P$: Random $x \leftarrow \mathbb{Z}_q$.

Prover:
- For $j = 1, \ldots, n$
  - Compute $f_j := \ell_j x + a_j$; $z_{a_j} = \tau_j x + s_j$; $z_{b_j} = \tau_j (x - f_j) + t_j$;
- Compute $z_d = (-t) x^n - \sum_{k=0}^{n-1} \rho_k x^k$;
- Compute $R := \sum_{i=0}^{N-1} (r_i \cdot x^n \cdot y^i) + \sum_{k=0}^{n-1} (R_k \cdot x^k)$;
- Compute $v_1 := s' + x \cdot \mathsf{sk}, v_2 := t' + x \cdot t$;
- $P \to V$: $(f_1, z_{a_1}, z_{b_1}, \ldots, f_n, z_{a_n}, z_{b_n}, z_d, R, v_1, v_2)$.

Verifier:
- For $i = 0, \ldots, N-1$, compute $c_i = \mathsf{pk}_i / c$.
- For all $j \in \{1, \ldots, n\}$, check $c_{\ell_j}^x c_{a_j} = \mathsf{Com}(f_j; z_{a_j})$ and $c_{\ell_j}^{x - f_j} c_{b_j} = \mathsf{Com}(0; z_{b_j})$;
- Check $\prod_{i=0}^{N-1} c_i^{\prod_{j=1}^{n} f_{j,i_j}} \cdot \prod_{k=0}^{n-1} c_{d_k}^{-x^k} = \mathsf{Com}(0; z_d)$, using $f_{j,1} = f_j$ and $f_{j,0} = x - f_j$;
- Check $\prod_{i=0}^{N-1} \left((E_i)^{x^n} \cdot [\![- \prod_{j=1}^{n} f_{j,i_j}; 0]\!] \right)^{y^i} \cdot \prod_{k=0}^{n-1} (D_k)^{x^k} = [\![0; R]\!]$, using $f_{j,1} = f_j$ and $f_{j,0} = x - f_j$.
- Check $g^{v_1} \mathsf{ck}^{v_2} = m \cdot c^x$;
- Output 1 iff all the checks pass.

Figure 3: ZK protocol for relation $\mathcal{R}$.

To prove that the protocol is *sound*, suppose the adversary creates $n+1$ accepting responses $f_1^{(0)}, \ldots, v_2^{(0)}, \ldots, f_1^{(n)}, \ldots, v_2^{(n)}$ to $n+1$ different challenges $x^{(0)}, \ldots, x^{(n)}$ on the same initial message $c, \ldots, m$.

We first show that $\ell_j \in \{0, 1\}$ for $j \in [1, n]$. Pick two responses $f_j^{(0)}, z_{a_j}^{(0)}, z_{b_j}^{(0)}$ and $f_j^{(1)}, z_{a_j}^{(1)}, z_{b_j}^{(1)}$ to challenges $x^{(0)}, x^{(1)}$ on the commitments c_{a_j}, c_{b_j}. By combining the verification equations we obtain $c_{\ell_j}^{x^{(0)} - x^{(1)}} = \mathsf{Com}(f_j^{(0)} - f_j^{(1)}; z_{a_j}^{(0)} - z_{a_j}^{(1)})$ and $c_{\ell_j}^{x^{(0)} - f_j^{(0)} - x^{(1)} + f_j^{(1)}} = \mathsf{Com}(0; z_{b_j}^{(0)} - z_{b_j}^{(1)})$. Defining $\ell_j = \frac{f_j^{(0)} - f_j^{(1)}}{x^{(0)} - x^{(1)}}$ and $\gamma_j = \frac{z_{a_j}^{(0)} - z_{a_j}^{(1)}}{x^{(0)} - x^{(1)}}$ we extract an opening of $c_{\ell_j} = \mathsf{Com}(\ell_j; \gamma_j)$. Furthermore, since $c_{\ell_j}^{x^{(0)} - f_j^{(0)} - x^{(1)} + f_j^{(1)}} = c_{\ell_j}^{(1-\ell_j)(x^{(0)} - x^{(1)})} = \mathsf{Com}(\ell_j(1-\ell_j)(x^{(0)} - x^{(1)}); \gamma_j(1-\ell_j)(x^{(0)} - x^{(1)})) = \mathsf{Com}(0; z_{b_j}^{(0)} - z_{b_j}^{(1)})$, either $\ell_j(1-\ell_j) = 0$ or the binding property of Pedersen commitment is broken. Thus, we have $\ell_j \in \{0, 1\}$ and extract $\ell = \ell_1 \ldots \ell_n$.

Then, the soundness is two-fold. In the first part, we prove $c = \mathsf{Com}(\mathsf{sk}) \wedge pk_\ell = g^{\mathsf{sk}}$ and extract sk. In the second part, we prove that $E_\ell = [\![1; r_\ell]\!] \wedge E_i = [\![0; r_i]\!], i \neq \ell$.

Let a_j be the number committed in c_{a_j}, from the verification equation $c_{\ell_j}^x c_{a_j} = \mathsf{Com}(f_j; z_{a_j})$ we conclude that $f_j^{(0)} = \ell_j x^{(0)} + a_j, \ldots, f_j^{(n)} = \ell_j x^{(n)} + a_j$ for all $j = 1, \ldots, n$ unless the adversary breaks the binding property of Pedersen commitment.

From the form of f_j's we have $f_{j,1} = \ell_j x + a_j$ and $f_{j,0} = (1 - \ell_j)x - a_j$. For $i \neq \ell$, it follows that $p_i(x) = \prod_{j=1}^{n} f_{j,i_j}$ is a polynomial of degree at most $n-1$, and for $i = \ell$ it is a polynomial of the form $p_\ell(x) = x^n + \ldots$. Therefore we can rewrite $\prod_{i=0}^{N-1} c_i^{\prod_{j=1}^{n} f_{j,i_j}} \cdot \prod_{k=0}^{n-1} c_{d_k}^{-x^k} = \mathsf{Com}(0; z_d)$ as

$$c_\ell^{x^n} \cdot \prod_{k=0}^{n-1} c_{*_k}^{x^k} = \mathsf{Com}(0; z_d), \tag{3}$$

for some fixed $c_{*_0}, \ldots, c_{*_{n-1}}$, which can be computed from commitments in $\{c_i\}_{i \in [0, N-1]}$ and the initial message.

Observe that the vectors $(1, x^{(\beta)}, \ldots, (x^{(\beta)})^n)$ can be viewed as rows in a Vandermonde matrix because $x^{(0)}, \ldots, x^{(n)}$ are all different. The matrix is invertible and we can therefore find a linear combination $(\alpha_0, \ldots, \alpha_n)$ of the rows that give us the vector $(0, \ldots, 0, 1)$. Combining the $n+1$ accepting verification equations, it follows that

$$c_\ell = \prod_{\beta=0}^{n} (c_\ell^{(x^{(\beta)})^n} \cdot \prod_{k=0}^{n-1} c_{*_k}^{(x^{(\beta)})^k})^{\alpha_\beta} = \mathsf{Com}(0; \sum_{\beta=0}^{n} \alpha_\beta z_d^{(\beta)}). \tag{4}$$

This equation gives us an extracted opening of c_ℓ to 0. Since $c_i = \mathsf{pk}_i / c$, and denoting $\mathsf{pk}_\ell = g^{\mathsf{sk}}$, we have $c = g^{\mathsf{sk}} \mathsf{ck}^t$, where $t = -\sum_{\beta=0}^{n} \alpha_\beta z_d^{(\beta)}$.

Then, by $g^{v_1} \mathsf{ck}^{v_2} = m \cdot c^x$ we have that

$$c = g^{(v_1^{(0)} - v_1^{(1)})(x^{(0)} - x^{(1)})^{-1}} \mathsf{ck}^{(v_2^{(0)} - v_2^{(1)})(x^{(0)} - x^{(1)})^{-1}}. \tag{5}$$

This equation extracts $\mathsf{sk} = (v_1^{(0)} - v_1^{(1)})(x^{(0)} - x^{(1)})^{-1}$.

Next, we start to prove that $E_\ell = [\![1; r_\ell]\!] \wedge E_i = [\![0; r_i]\!], i \neq \ell$. Denote $E_i = [\![e_i; r_i]\!]$. Since x is randomly chosen after the D_k's are committed, by the verification equation $\prod_{i=0}^{N-1} \left((E_i)^{x^n} \cdot [\![- \prod_{j=1}^{n} f_{j,i_j}; 0]\!] \right)^{y^i} \cdot \prod_{k=0}^{n-1} (D_k)^{x^k} = [\![0; R]\!]$, we have that $\prod_{i=0}^{N-1} \left((E_i)^{x^n} \cdot [\![- \prod_{j=1}^{n} f_{j,i_j}; 0]\!] \right)^{y^i} \cdot \prod_{k=0}^{n-1} (D_k)^{x^k}$ encrypts a zero polynimial w.r.t x with overwhelming probability (by the Schwartz-Zippel Lemma). Therefore, by denoting $i_{j,1} = \ell_j, i_{j,0} = 1 - \ell_j$, we obtain $Q(y) = \sum_{i=0}^{N-1} (e_i - \prod_{j=1}^{n} i_{j,i_j}) \cdot y^i = 0$. Since y is randomly chosen after the E_i's are encrypted, $Q(y)$ is a zero polynomial w.r.t y with overwhelming probability (by the Schwartz-Zippel Lemma). Hence, we have $e_i = \prod_{j=1}^{n} i_{j,i_j}$ for $i \in [0, N-1]$.

To prove that the protocol is *special honest verifier SK*, we build a simulator that is given $x, y \leftarrow \mathbb{Z}_q$. It first randomly picks $f_1, \ldots, v_2 \leftarrow$

$\mathbb{Z}_q$. It then picks $c \leftarrow \mathbb{G}$ at random and $c_{\ell_1}, \ldots, c_{\ell_n}, c_{d_1}, \ldots, c_{d_{n-1}} \leftarrow \mathsf{Com}(0)$ as random commitments to 0. Next, it picks $U_i, R_i \leftarrow \mathbb{Z}_q$ at random and computes $D_i := [\![U_i; R_i]\!]$ for $i \in [1, n-1]$. After the random selection, it computes $c_i := \mathsf{pk}_i / c;\ c_{a_j} := c_{\ell_j}^{-x} \mathsf{Com}(f_j; z_{a_j})$, $c_{b_j} := c_{\ell_j}^{f_j - x} \mathsf{Com}(0; z_{b_j}),\ m := g^{v_1} h^{v_2} c^{-x}$, and

$$c_{d_0} := \prod_{i=0}^{N-1} c_i^{\prod_{j=1}^{n} f_{j,i_j}} \cdot \prod_{k=1}^{n-1} c_{d_k}^{-x^k} \cdot \mathsf{Com}(0; -z_d) \tag{6}$$

and

$$D_0 := \frac{[\![0; R]\!]}{\prod_{i=0}^{N-1} \left((E_i)^{x^n} [\![- \prod_{j=1}^{n} f_{j,i_j}; 0]\!] \right)^{y^i} \cdot \prod_{k=1}^{n-1} (D_k)^{x^k}}. \tag{7}$$

The simulator outputs the transcript $(y, c, \ldots, m, x, f_1, \ldots, v_2)$.

We argue that the adversary cannot distinguish the simulation from a real argument. First, in both real proofs and simulated proofs, $f_1, \ldots, v_2$ are uniformly random in $\mathbb{Z}_q$; c is uniformly random in $\mathbb{G}$. Furthermore, by the verification equations, $c_{a_1}, c_{b_1}, \ldots, c_{a_n}, c_{b_n}, m$, c_{d_0}, D_0 are determined by $f_1, \ldots, v_2$ and $c, c_{\ell_1}, \ldots, c_{\ell_n}, c_{d_1}, \ldots, c_{d_{n-1}}$, $D_1, \ldots, D_{n-1}$ both in real and in simulated proofs. The adversary's advantage must come from being able to distinguish $c, c_{\ell_1}, \ldots, c_{\ell_n}, c_{d_1}$, $\ldots, c_{d_{n-1}}, D_1, \ldots, D_{n-1}$ in real and simulated proofs. To do so, the adversary must either break the binding property of Pedersen commitment or break the IND-CPA property of ElGamal encryption by a standard hybrid argument. □

6 DESIGN AND CLIENT INTERFACES

Building on our system architecture (Section 3.4), we now explain our design and client interfaces. Section 7 describes our implementation.

Design elements. VoteXX differs from other election systems in that the BB is at the center. The BB receives all posts through an ACS; all other communication is directly peer-to-peer, or in person. The BB, via the ACS, is part of a public, preexisting decentralized infrastructure. The BB uses a multicast feature of the ACS, allowing all BB instances, auditors, and other observers to record the same data sent through the network at the same time.

Client interfaces. There are 4 clients, a voter client, a hedgehog client, an EA client, and an Auditor client. Each voter client operates like a calculator, without state or persistent storage. The voter can enter their YES or NO passphrase on another voter client device at any time to regenerate their ballot, and the voter client will verify that the ballot is properly posted to the BB.

The hedgehog client is integrated into the same mobile phone app as the voter client. Any voter can be a hedgehog for themselves or other voters. Voters and hedgehogs can send and receive ballot secrets directly between each other using their ACS identities.

The EA client posts data associated with EA operations, such as starting an election by posting a signed election parameters file.

Any client can read and write messages to the BB. Messages are ignored unless they are signed by eligible clients. Auditor clients read data posted directly from the ACS to the BB; they verify signatures and validate posted data.

7 IMPLEMENTATION

We built proof-of-concept implementations for all components of VoteXX, included as an artifact in our submission. We wrote the EA and auditor in Java with the BouncyCastle [31] library. We wrote the nullification and proofs in C++ with the cryptopp [12] library. Both implementations use the secp256k1 group. We can send inputs and outputs through ACS clients written in Golang using the native cryptographic libraries to interact with the BB, which currently is a simple file store utility.

We benchmarked specific operations on a PC using a AMD Ryzen 5 5600X 6-Core Processor with 2 16 GiB DIMMs at 2133 MT/s. The most expensive voting operation, tallying, took 15.34 seconds for a simulated 2^{20} (1 million) voters. Assuming that a very high 2^{17} (25%) of voters nullify, it would take 9.21 minutes to verify the proofs. Each proof was 5.93 KB in size. See Section 8.4 for more general performance analysis.

8 DISCUSSION

We now discuss our major design decisions, nullification options (cancel or flip), not revealing nullifications, performance analysis, extensions, and open problems.

8.1 Major Design Decisions

Toward our goal of addressing improper influence and supporting online verifiable elections, we made three major design decisions: (1) Nullification achieves the theoretically optimal *coercion resistance*, and using hedgehogs depends on a more realistic assumption than that assumed in previous work. (2) Our decentralized architecture provides *availability* and *malware resistance*. (3) In-person registration involving passphrases enhances voter authentication and supports key functionality for *malware resistance*.

Nullification and hedgehogs. Nullification allows the voter to share a passphrase anytime after they conceive of it. In-person registration ensures the voter knows their passphrases, providing ample opportunity even for captive voters (*e.g.,* a spouse or child) to signal a hedgehog. Because each passphrase can nullify a ballot only in one direction (the NO key can only vote NO or nullify YES; the YES key can only vote YES or nullify NO), voter intent matters and a signal to coordinate with a hedgehog can be optional. For example, a candidate who is a hedgehog might always nullify a ballot cast against them if possible.

Decentralized architecture. Routing all audit data through the ACS creates a special challenge to the adversary not present in traditional election systems: Any attack on the infrastructure must disable a much larger system, where there is an independent financial incentive for it to remain online. The BB, decentralized through the ACS, is not vulnerable to denial-of-service. Flooding the BB with data [29] is limited as adversaries must pay for ACS bandwidth. Because all BB data are public and we use known E2E-voting constructions, the system meets the requirements for voter verifiable ballots, contestability, and auditability Park et al. [36].

In-Person registration. Our registration design roots trust into passphrases known to the voter and written on physical paper associated with a specific person. This design provides a critical feature for the system's *malware resistance*: passphrases make it possible to detect and prove misbehavior by the software because

all data posted to the BB can be regenerated with the passphrases on any device.

VoteXX greatly complicates undetectable wholesale attacks: the adversary must deploy malicious software across all devices controlled by checking with a passphrase. The deployment must go undetected forever, or at least until the election completes. If the attack is detected after the election, the adversary risks loss of confidence from a provably improper election outcome. It would be intractable for an adversary to remain undetected for a useful period of time. Our design decisions allow VoteXX to prevent undetectable wholesale attacks at scale and provide detection and mitigation against retail attacks.

8.2 Cancel or Flip

Our design supports a variety of options for implementing the semantics of nullification, including what we call "cancel" or "flip." We recommend flip, which is the default. Consider a vote that might have been nullified by one or more entities. We will describe the case for when there are two ballot choices (See Section 8.5 for the general case of k ballot choices). Assume that this vote selects from one of two ballot choices numbered 0, 1. With *cancel*, the vote is cancelled if and only if at least one entity nullified it (and this idea can be generalized to at least t entities for some threshold t). With *flip*, the vote becomes $x + y \mod 2$, where x is the ballot choice of the vote, and y is the number of times the vote was nullified. Intuitively, cancel gives the voter the ability to cancel the vote, whereas flip gives the voter the ability to randomize the vote.

Each of these options can be implemented using different algebraic operations during Step 1 of the third phase (Final Tally) of Protocol 3: `AND` for cancel (realized with a homomorphic addition of the encrypted flags for each ballot followed by a plaintext equality test with $[\![0]\!]$), and `ADDITION` modulo 2 for flip (realized with mix and match. Step 2 replaces the final summation with a verifiable shuffle and threshold decryption of the flag set for each key).

A useful application of flip arises for a common form of low-intensity coercion. Suppose during remote voting at home, a coercer tells their spouse to vote for Alice and watches them comply, but the coercer does not collect the spouse's keys. Without any advance planning, the spouse can later flip their vote to Bob without the coercer knowing.

8.3 Not Revealing Which Ballots are Nullified

Our base proposal irrevocably hides whether any particular ballot was nullified. This action provides absolute protection of voters from coercers. If a voter's keys are exposed (e.g., malware or sharing a password), it puts the voter in the same position as a potential coercer: they cannot know if the ballot was nullified, because they do not know what the other parties with access to their keys might have done with respect to nullification.

A facility that could be helpful in this situation would let a voter view any nullification of their vote but only in a special booth at a controlled location. An extension to our current protocol could allow for such a change, and can be designed so that the total number of voters for which such viewings can be arranged is at least public. Any public process creates a cost asymmetry for an attacker to force each voter through the process to complete their coercion. Assuming a legal and policy framework that provides protection for voters is in place, such solutions achieve our stated goals.

8.4 Performance Analysis

We analyze the running time of VoteXX for elections with T trustees, V voters, and H hedgehogs. If a passphrase is ℓ characters long with α possible characters, registration setup takes $\Theta(\ell\alpha T)$ work (comprised of modular exponentiations and Σ−Protocols). The proof size and verification time for the auditor is also $\Theta(\ell\alpha T)$. Example parameters might be $\alpha = 64$ characters of length $\ell = 20$ and $T = 10$ trustees. The shuffle proof dominates registration, generally taking $\Theta(VT \log V)$. Each vote has a constant amount of signatures, encryptions, and Σ−Protocols for the voter. Proof size and verification time for the auditor is $\Theta(V)$. The provisional tally consists of another shuffle, $\Theta(VT \log V)$, and decryption (subsumed in the shuffle), with the proof size and verification time of the same order for the auditor.

Nullification is an involved protocol. As mentioned in Section 4, to avoid a quadratic bottleneck during the final tally, it is essential to process hedgehog flags as they arrive. Each hedgehog performs $\Theta(V)$ work (encryptions and Σ−Protocols) that an auditor must fetch and validate (space and time of $\Theta(V)$). For each of the V flags from one hedgehog, the trustees can precompute a logic gate (two-input gates are effectively constant time). Applying the gate to the inputs is $\Theta(T)$ (plaintext equality tests and Σ−Protocols). In total, nullification is $\Theta(HVT)$ work for the EA and auditors, with same order proof size on the BB. The final tally is fast: $\Theta(VT)$ work (consisting of decryption and Σ−Protocols) for the trustees and auditors, with same order proof size.

8.5 Extensions

We briefly describe several possible extensions of VoteXX.

Multiple candidates. VoteXX can be easily extended to support an election with multiple candidates. For example, for a k-candidate race, the voter can register k key pairs and then vote using the desired key. Without any major changes, nullification still operates as before. For example, to perform a flip, the system can use an addition modulo k to determine what flip to apply to the initially cast vote. Since the nullification protocol scales linearly in the number of voters and hedgehogs, introducing multiple candidates does not affect the overall performance of the nullification process.

Voting in person or by mail. To support the existing voting infrastructure, VoteXX can allow for a setting where the voting is accomplished by mail or in precincts using paper ballots. This capability can be achieved by incorporating a code-voting protocol, such as that used in Remotegrity [46].

Malware protection. To enhance protection against malware, where the voting device is running malicious software and can alter the operations performed by the voter, VoteXX allows for a two-phase voting process. In Phase 1, the user submits a vote or a vote commitment. In Phase 2, using a different device, the voter checks if the submission is correctly posted on the BB. Optionally, this extension can include an additional set of keys, where the user submits a payload signed with the additional keys and thereby "*locks in*" their submission.

Roster changes. If detected early in the election, it is possible to contest and remove a compromised passphrase. Providing proper documentation, the affected voter would rerun the in-person registration process.

Online registration. For lower-security elections, it is possible to replace the in-person registration with an online registration that follows appropriate identification mechanisms or uses an identity verification service [44].

8.6 Open Problems and Future work

Open problems include:

(1) Explore how the number of nullifications might provide a measure of coercion [20]. This measure might even be used to reject an election outcome, if there were too many nullifications. A danger is that such a mechanism might be abused to discredit a valid outcome.

(2) Investigate how our technical machinery of nullification might be used in non-voting applications, such as contract signing.

Next steps for VoteXX include the following. Building on our UC proofs (Appendix C), we plan to carry out a formal-methods analysis of selected VoteXX protocols using protocol-analysis tools and build a formally verified implementation of key system elements. We also plan to conduct a pilot election and user study to assess the overall usability and how well VoteXX achieved its design goal of providing a voting experience that is intuitive with few steps. Results can help us improve the system and facilitate widespread adoption. To enhance the availability of VoteXX, we plan to decentralize the protocol further, enabling a subset of the EA to perform certain election steps.

9 CONCLUSION

Leveraging hedgehogs, an ACS, BBs, and user-generated passphrases, VoteXX provides a new, practical, and versatile solution to improper influence in elections against strong adversaries who learn the voter's voting keys. VoteXX works through the use of nullification supported by voter associates whom we call hedgehogs. In comparison with previous approaches, our solution makes fewer assumptions and protects against stronger adversaries. By separating our mechanism for mitigating improper influence from the mechanisms of ballot marking and collection, our technique works with a wide range of voting systems, including precinct voting with paper ballots, voting by mail, and Internet voting. For example, our mechanism works harmoniously with techniques for mitigating malware attacks, including allowing voters to check across multiple systems and devices. Also, our nullification mechanism can be used in addition to other mechanisms for mitigating improper influence.

Currently, election systems without voting booths are vulnerable to potential improper influence attacks. For example, a nation state, terrorist organization, billionaire, or anonymous hackers might offer significant amounts of money to vote for certain candidates. It could likely be impossible to know the extent to which such attacks succeeded. Such attacks would discredit the election, and re-running the election with the same technology would not resolve the issue. Our paper offers a solution to this threat that achieves the theoretically best possible result. Having demonstrated that extreme coercion resistance is possible, even in Internet voting, democratic societies should insist that, as a matter of due diligence, all voting systems should provide coercion resistance. Our work protects voting beyond the booth, and such voting is an essential enabler for the advance of democracy.

ACKNOWLEDGMENTS

This project was supported in part by xx network. Clark was supported in part by NSERC and Raymond Chabot Grant Thornton under grants IRCPJ/545498-2018 and RGPIN/04019-2021. Preneel was supported in part by CyberSecurity Research Flanders with reference number VR20192203. Sherman was supported in part by the National Science Foundation under SFS grants DGE-1241576, 1753681, and 1819521, and by the U.S. Department of Defense under CySP grants H98230-17-1-0387, H98230-18-1-0321, H98230-19-1-0308, and H98230-20-1-0384.

REFERENCES

[1] 2022. REDACTED (extended abstract) for anonymous submission.
[2] Ben Adida. 2008. Helios: Web-Based open-audit voting. In *USENIX Security Symposium*. 335–348.
[3] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. 2015. Incoercible multi-party computation and universally composable receipt-free voting. In *Annual Cryptology Conference*. Springer, 763–780.
[4] Roberto Araujo, Sebastien Foulle, and Jacques Traoré. 2010. A practical and secure coercion-resistant scheme for Internet voting. *Toward Trustworthy Elections* LNCS 6000 (2010).
[5] Roberto Araujo, Narjes Ben Rajeb, Riadh Robbana, Jacques Traoré, and Souheib Yousfi. 2010. Towards Practical and Secure Coercion-Resistant Electronic Elections. In *CANS*.
[6] David Bernhard, Olivier Pereira, and Bogdan Warinschi. 2012. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *ASIACRYPT*.
[7] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 136–145.
[8] David Chaum. 2012. Random-Sample Voting. (2012). Online.
[9] David Chaum and Torben Pryds Pedersen. 1992. Wallet Databases with Observers. In *CRYPTO*.
[10] Jeremy Clark and Urs Hengartner. 2011. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *Financial Cryptography*.
[11] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. 2008. Civitas: Toward a Secure Voting System. In *IEEE Symposium on Security and Privacy*. 354–368.
[12] Community Maintained, Originally written by Wei Dai. 1995. Crypto++ Library 8.7. https://www.cryptopp.com/ Accessed: 2023-03-08.
[13] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 1994. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO*.
[14] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*.
[15] Stéphanie Delaune, Steve Kremer, and Mark Ryan. 2006. Coercion-Resistance and receipt-freeness in electronic voting. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*. IEEE, 12+.
[16] Aleksander Essex, Jeremy Clark, and Urs Hengartner. 2012. Cobra: Toward Concurrent Ballot Authorization for Internet Voting. In *EVT/WOTE*.
[17] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*. 186–194.
[18] Amos Fiat and Adi Shamir. 1990. Witness Indistinguishable and Witness Hiding Protocols. In *ACM STOC*.
[19] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. 1992. A practical secret voting scheme for large scale elections. In *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 244–251.
[20] Gurchetan S Grewal, Mark D Ryan, Sergiu Bursuc, and Peter YA Ryan. 2013. Caveat Coercitor: Coercion-evidence in electronic voting. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 367–381.
[21] Jens Groth and Markulf Kohlweiss. 2015. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Springer, 253–280.
[22] Thomas Haines, Johannes Müller, and Iñigo Querejeta-Azurmendi. 2023. Scalable Coercion-Resistant E-Voting under Weaker Trust Assumptions. In *ACM SAC*.

[23] Carmit Hazay and Yehuda Lindell. 2010. *Efficient Secure Two-Party Protocols.* Springer.
[24] Martin Hirt and Kazue Sako. 2000. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *EUROCRYPT*.
[25] I2P. 2003. Invisible Internet Project. https://www.geti2p.net. Accessed on 01-05-2022.
[26] Markus Jakobsson and Ari Juels. 2000. Mix and Match: Secure function evaluation via ciphertexts. In *ASIACRYPT*.
[27] Ari Juels, Dario Catalano, and Markus Jacobsson. 2005. Coercion-Resistant electronic elections. In *ACM WPES*.
[28] Mahimna Kelkar, Kushal Babel, Philip Daian, James Austgen, Vitalik Buterin, and Ari Juels. 2023. Complete Knowledge: Preventing Encumbrance of Cryptographic Secrets. Cryptology ePrint Archive, Paper 2023/044. https://eprint.iacr.org/2023/044 https://eprint.iacr.org/2023/044.
[29] Reto Koenig, Rolf Haenni, and Stephan Fischli. 2011. Preventing Board Flooding Attacks in Coercion-Resistant Electronic Voting Schemes. In *SEC*.
[30] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2010. Accountability: Definition and Relationship to Verifiability. In *ACM CCS*.
[31] Legion of the Bouncy Castle. 2000. Bouncy Castle crypto APIs. https://www.bouncycastle.org/java.html Accessed: 2022-05-05.
[32] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. 2020. VoteAgain: A scalable coercion-resistant voting system. In *USENIX Security*.
[33] Tal Moran and Moni Naor. 2006. Receipt-Free Universally-Verifiable Voting With Everlasting Privacy. In *CRYPTO*.
[34] T Okamoto. 1992. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*.
[35] Oxen. 2020. Privacy made simple. https://www.oxen.io. Accessed on 01-05-2022.
[36] Sunoo Park, Michael Specter, Neha Narula, and Ronald L. Rivest. 2020. Going from Bad to Worse: From Internet Voting to Blockchain Voting. (2020). Online.
[37] Torben Pryds Pedersen. 1991. A threshold cryptosystem without a trusted party. In *EUROCRYPT*.
[38] C P Schnorr. 1991. Efficient Signature Generation by Smart Cards. *Journal of Cryptography* 4 (1991), 161–174.
[39] Ben Smyth. 2019. Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822.
[40] Oliver Spycher, Rolf Haenni, and Eric Dubuis. 2010. Coercion-Resistant Hybrid Voting Systems. In *EVOTE*.
[41] TOR. 2002. The TOR Project. https://www.torproject.org. Accessed on 01-05-2022.
[42] M. Volkamer and R. Grimm. 2006. Multiple Casts in Online Voting: Analyzing Chances. In *EVOTE*.
[43] R Wen and R Buckland. 2009. Masked Ballot Voting for Receipt-Free Online Elections. In *VOTE-ID*.
[44] Wikipedia. 2022. Identity Verification Service. https://en.wikipedia.org/wiki/Identity_verification_service Accessed: 2022-05-07.
[45] xx.network. 2021. A Quantum Leap in privacy. https://www.xx.network. Accessed on 01-05-2022.
[46] Filip Zagórski, Richard T. Carback III, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. 2013. Remotegrity: Design and Use of an End-to-End Verifiable Remote Voting System. In *ACNS*.
[47] Bingsheng Zhang, Roman Oliynykov, and Hamed Balogun. 2019. A Treasury System for Cryptocurrencies: Enabling Better Collaborative Intelligence. In *The Network and Distributed System Security Symposium 2019*.

A ACRONYMS AND ABBREVIATIONS

ACS	anonymous communication system
BB	bulletin board
CDS	stacked through conjunction and disjunction
DDH	decisional Diffie-Hellman
DKG	distributed key generation
EA	Election Authority
E2E	end-to-end
NIZK	non-interactive zero knowledge
TA	Tally Authority
UC	universal composability
VA	Voting Authority
ZK	zero knowledge
ZKP	zero-knowledge proof

B EXPLANATION OF PROPERTIES IN TABLE 1

(0) *System resists coercion when the influencer: acts before/during registration.* In a number of coercion resistance mechanisms, the voter is expected to register a user-chosen key, password, or be assigned a key. If registration can be done in the presence of the adversary or using inputs supplied by the adversary (and compliance can be checked by the adversary), without impacting coercion resistance, we award ●. Otherwise if the system assumes the adversary cannot be active during this process or places limitations on their actions in corrupting the registrants, we award ○.

(1) *System resists coercion when the influencer: colludes with the EA.* A system should maintain coercion-resistance even when the coercer is able to corrupt a minority of the EA (◕). Limitations on this assumption might result in ◐: for example, in fake credentials, it is assumed the coercer can corrupt any member of the EA but the voter must know which EA member has not been corrupted. Other systems fail to provide coercion resistance (○) under this assumption.

(2) *System resists coercion when the influencer: colludes with hardware manufactures.* A system that does not rely on trusted hardware to provide coercion-resistance is awarded ●. By contrast, as system that makes hardware assumptions beyond typical computational equipment, such as a trusted execution environment [3], is awarded ○ (along with systems that do not provide coercion resistance).

(3) *System resists coercion when the influencer: acts at any time.* Assumming an influencer cannot act at all times (see Property 5), are there additional restrictions on when they can act? If not: ●. Systems receive ○ include ones that assume the influencer does not act before or during registration, and systems like re-voting that assume the coercer does not act at the very end of the voting period (blocking a re-vote).

(4) *System resists coercion when the influencer: learns all information stored by the voter, including all keys required by the protocol.* Our main contribution is that VoteXX achieves coercion resistance even if all the voter's stored keys/secrets are leaked—extreme coercion resistance. This property is stronger than the literature, which generally assumes voters can establish and maintain secret keys or passwords (and lie convincingly about them as necessary) that will need to be recalled to cast a vote: ○. While non-verifiable re-voting does not require voter secrets or private keys, end-to-end verifiable systems do as a way to cryptographically link ballots and prevent multiple votes from the same voter. VoteAgain is designed as an exception to this rules; in it, such keys exist but are maintained by a special election trustee so voters do not need to. However it must be completely trusted for coercion resistance (and in fact, must be trusted for ballot privacy and election integrity as well [22]). Use of a trusted third party also receives ○. Trusting a hardware enclave to maintain keys is awarded ◐. Systems that assume the influencer cannot impersonate the voter but do not provide a specific mechanism for online settings are also awarded ○.

(5) *System resists coercion when the influencer: learns every action taken by the voter.* With reasonable assumptions on how voting works, this property is in fact shown to be impossible to achieve, as the voter can never act independently [24]. We include it to highlight this fact and as an open problem: perhaps some other trust model or assumptions on the voter would enable this property.
(6) *Voter can undo coercion undetectably.* Coercion needs to be corrected when the voter's intent is different than the influencer's. Assuming the coercion resistance mechanism is allowed to work, if the voter is always able to vote their true intent, the mechanism is awarded ●. By contrast, systems are awarded ○ when voters cannot reliably vote their true intent. These systems, however, can still be considered coercion resistant if they do allow the voter to cancel the coercer's intent by spoiling, nullifying, or randomizing their ballot. In VoteXX, voters can vote their true intent if they can predict the influencer's actions and respond strategically. However, we cannot assume this ability will always be the case, and so, at best, voters can cancel or randomize their ballots. In VoteXX this choice depends on a system configuration discussed in Section 8.2.
(7) *System is inexpensive.* A system that does not introduce new expenses beyond the EA running a server and voters having access to standard computational devices is awarded ●. A system that requires special equipment or hardware for either the EA or for the voters is awarded ○ (*e.g.,* special hardware for digital signatures [3]).
(8) *System has low cognitive burden.* If the human voters strategy for evading coercion is automated or does not require any cognitive effort, it is awarded ●. If the voter needs to remember passwords, it is awarded ◐. Any strenuous mental effort (*e.g.,* remembering an integer offset and performing mental arithmetic [43]) is awarded ○.
(9) *System has security proof.* A universal composability proof is awarded ●, a game-based security definition and proof is awarded ◐, while informal security arguments and sketches are awarded ○.

C SECURITY ANALYSIS

We formally state and prove properties of VoteXX in the *Universal Composability (UC)* framework [7]. To begin, we define types of coercion and state the security of VoteXX. Next, we give a UC specification of VoteXX and state and prove a theorem that characterizes its security properties, using "cancel" nullification (see Section 8.2). We model our proofs in part from those of Alwen, Ostrovsky, Zhou, and Zikas [3].

C.1 Preliminaries

In this section, we formally define the cryptographic primitives in VoteXX and their properties.

NIZK. A *non-interactive zero-knowledge proof (NIZK)* for relation $\mathcal{R}$ consists of four probabilistic polynomial algorithms (Setup, Prove, Verify, Sim) such that

$(\sigma, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$: The setup algorithm outputs a common reference string σ and a simulation trapdoor τ for relation $\mathcal{R}$.

$\pi \leftarrow \mathsf{Prove}(\mathcal{R}, \sigma, \phi, w)$: the prover algorithm takes as input a common reference string σ and $(\phi, w) \in \mathcal{R}$ and outputs a proof π.

$0/1 \leftarrow \mathsf{Verify}(\mathcal{R}, \sigma, \phi, \pi)$: the verification algorithm takes as input a common reference string σ, a statement ϕ and a proof π, and it returns 0 OR 1 for rejection OR acceptance, respectively.

$\pi \leftarrow \mathsf{Sim}(\mathcal{R}, \tau, \phi)$: the simulation algorithm takes as input a simulation trapdoor τ and a statement ϕ, and it outputs a proof π.

Completeness. Completeness says that an honest prover can always convince an honest verifier. Formally, for all $(\phi, w) \in \mathcal{R}$,

$$\Pr[(\sigma, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}); \pi \leftarrow \mathsf{Prove}(\mathcal{R}, \sigma, \phi, w) : \mathsf{Verify}(\mathcal{R}, \sigma, \phi, \pi) = 1] = 1.$$

Zero-Knowledge. A *proof is zero-knowledge* if it does not leak any information except that the statement is true. Consider the following experiment:

Experiment $\mathrm{EXPT}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$:

(1) For a relation $\mathcal{R}$, $(\sigma, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$, $(\phi, w) \in \mathcal{R}$, the challenger computes $\pi_0 \leftarrow \mathsf{Prove}(\mathcal{R}, \sigma, \phi, w)$ and $\pi_1 \leftarrow \mathsf{Sim}(\mathcal{R}, \tau, \phi)$.
(2) The challenger picks a random bit $b \in \{0, 1\}$.
(3) $\mathcal{A}$ is given (σ, π_b) as input, and it outputs a guess bit $b' \in \{0, 1\}$.
(4) If $b = b'$, output 1; otherwise, output 0.

A *NIZK is zero-knowledge* if the adversary $\mathcal{A}$'s advantage $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}}(\mathcal{A}, \lambda) := |2 \cdot \Pr[\mathrm{EXPT}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = 1] - 1|$ is negligible in λ.

Soundness. Soundness says that a prover cannot prove a false statement. Consider the following experiment:

Experiment $\mathrm{EXPT}^{\mathsf{sound}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$:

(1) For a relation $\mathcal{R}$, $(\sigma, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$.
(2) Given σ as input, $\mathcal{A}$ outputs (ϕ, π).
(3) If $\mathsf{Verify}(\mathcal{R}, \sigma, \phi, \pi) = 1$ and $\phi \notin L_{\mathcal{R}}$, output 1; otherwise, output 0.

A *NIZK is sound* if the adversary $\mathcal{A}$'s advantage $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{NIZK}}(\mathcal{A}, \lambda) := \Pr[\mathrm{EXPT}^{\mathsf{sound}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = 1]$ is negligible in λ.

Encryption scheme. An *encryption scheme* consists of three probabilistic polynomial algorithms (Keygen, Enc, Dec). We require the underlying encryption scheme to be *indistinguishable under chosen-plaintext attack (IND-CPA)*. Consider the following experiment:

Experiment $\mathrm{EXPT}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{Enc}}(\lambda)$:

(1) The challenger performs the key generation algorithm $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(\lambda)$ and sends pk to the adversary $\mathcal{A}$.
(2) $\mathcal{A}$ sends m_0, m_1 to the challenger.
(3) The challenger picks a random bit $b \in \{0, 1\}$ and sends $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$ to $\mathcal{A}$.
(4) $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$. If $b = b'$, output 1; otherwise, output 0.

An *encryption scheme is IND-CPA secure* if the adversary $\mathcal{A}$'s advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Enc}}(\mathcal{A}, \lambda) := |2 \cdot \Pr[\mathrm{EXPT}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{Enc}}(\lambda) = 1] - 1|$ is negligible in λ.

Signature. A *signature scheme* consists of three probabilistic polynomial algorithms (Keygen, Sign, Verify). We require the underlying signature scheme to be *existentially unforgeable under chosen-message attack (EUF-CMA)*. Consider the following experiment:

Experiment $\mathrm{EXPT}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sig}}(\lambda)$:

(1) The challenger performs the key generation algorithm $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(\lambda)$ and sends pk to the adversary $\mathcal{A}$.
(2) $\mathcal{A}$ can repeatedly request for signatures on chosen messages $(m_0, \ldots, m_q)$, and receives the valid signatures $(\sigma_0, \ldots, \sigma_q)$ in response.
(3) $\mathcal{A}$ outputs a message and signature (m^*, σ^*).
(4) If m^* is not one of the messages requested in Step 2, and $\mathsf{Verify}_{\mathsf{pk}}(m^*, \sigma^*) = 1$, output 1; otherwise, output 0.

A *signature scheme is EUF-CMA* if the adversary $\mathcal{A}$'s advantage $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{Sig}}(\mathcal{A}, \lambda) := \Pr[\mathrm{EXPT}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{Sig}}(\lambda) = 1]$ is negligible in λ.

Bulletin Board. A *bulletin board* is required for any voting scheme to record ballots and other related information. We model the bulletin board as a UC global functionality $\mathcal{G}_{\mathsf{BB}}$, which is depicted in Fig. 4. It has two interfaces: Read and Write, and records all the valid messages. $\mathcal{G}_{\mathsf{BB}}$ ensures that all the communication with the bulletin board is anonymous.

Functionality $\mathcal{G}_{\mathsf{BB}}$

The ideal functionality $\mathcal{G}_{\mathsf{BB}}$ is globally available to all participants. It is parameterized with a predicate Validate.
Upon initialization, set Storage := $\emptyset$.
Upon receiving (Read, sid) from P:
- Let val := Storage[sid];
- Return (READ, sid, val) to the requestor.

Upon receiving (Write, sid, inp) from P, do the following:
- Let val := Storage[sid];
- If Validate(val, inp) = 1, then set Storage[sid] := val||inp, return (Receipt, sid) to the requestor;
- Otherwise, return (Reject, sid) to the requestor.

Figure 4: Functionality $\mathcal{G}_{\mathsf{BB}}$

C.2 Types of Coercion

To articulate the capability of the coercer in VoteXX, we define a new type of coercion: *extreme coercion*, which differs from previous notions of *semi-honest coercion (receipt-freeness)* and *active coercion*.

Semi-Honest coercion. This type of coercion is most common in the literature; resistance against semi-honest coercion is called receipt-freeness. In this case, the coercer provides an input to the coerced party and expects evidence that such input was used, which evidence is called a "*receipt*." For example, the receipt can be the entire view of the protocol execution.

Active coercion. Moran and Naor [33] proposed this stronger type of coercion. Instead of merely requiring a receipt, the coercer can query the current view interactively and send commands to the coerced party during the protocol execution.

Extreme coercion. We define this new stronger type of coercion, called "extreme coercion," which captures the real world more accurately. The coercer can obtain *all* the secret keys and passwords of the coerced party, and can perform operations in substitution for the coerced party. The coerced party, however, can secretly communicate with other people via some untappable channel.

Extreme coercion captures real world coercion more realistically because the coercer may ask the coerced party to hand over their device to extract the secret keys and monitor the coerced party's action. Because we consider it impossible to coerce a target throughout their entire life, they can recruit a hedgehog and agree on some secret action in advance.

C.3 Ballot privacy, coercion resistance and verifiability.

In this section, we give (informal) definitions of ballot privacy, coercion resistance, and verifiability. Then, we give intuition why VoteXX satisfies these properties. We formalize the secure definition in Section C.4 under the UC framework and argue that the UC definition implies these properties.

Ballot privacy [19]. All votes must be secret.

The link between the voter and the corresponding public key in the roster is hidden by the verifiable shuffle in the registration phase. In addition, all the ballots are encrypted under the EA's threshold encryption scheme in the voting phase. Thus, VoteXX ensures ballot privacy assuming that the majority of EA trustees are honest.

Coercion resistance. No coercer can tell if the coerced party is trying to deceive.

The ballots and nullification requests are posted on the BB via an ACS to avoid identity leakage. In the nullification phase, the flags marking which ballots are to be nullified are encrypted and a ZKP establishes knowledge of the corresponding secret key. In addition, we assume that there is an untappable channel between the voter and his hedgehog(s) that cannot be blocked by the coercer. Therefore, the coercer cannot stop a coerced party from nullifying his vote and cannot know if the ballot is nullified.

Verifiability [19]. No one can falsify the result of the voting.

We assume an honest BB and the messages posted on the BB cannot be deleted OR changed. In the provisional and final tallies, VoteXX uses ZKPs to ensure that the shuffle and decryption are performed correctly. The max-and-match SFE protocol [26] in the final tally is publicly verifiable.

C.4 Security Definition.

We define the security of VoteXX in the UC framework [7]. A protocol is represented as a set of *interactive Turing machines (ITMs)*, where each ITM represents the program to be run by a participant. There are two additional entities: the *environment* $\mathcal{Z}$ and the *adversary* $\mathcal{A}$. The environment $\mathcal{Z}$ can communicate with $\mathcal{A}$ and provides inputs to the parties. We assume that each ITM is a *probabilistic polynomial-time (PPT)* machine.

Security is based on the indistinguishability between *real/hybrid world executions* and *ideal world executions*. Specifically, in the ideal world, all the participants are dummy parties and there is an ideal functionality $\mathcal{F}$ that serves as a trusted third party. We say that a *protocol π UC-realizes $\mathcal{F}$* if and only if, for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish between the real/hybrid world and the ideal world.

Our protocol contains *pre-tally* and *nullification* phases. Therefore, by comparing the final tally with the pre-tally, the coercer can discern the amount by which nullification altered the results. The coercer, however, cannot attribute such difference to the voter. To capture this feature, we model the ideal world as follows.

Ideal deception. Our treatment of incoercibility in the ideal world is inspired by Alwen, Ostrovsky, Zhou, and Zikas [3]. Alwen et al. define an ideal deception strategy DI as a mapping applied on the message given by the coercer to an intended choice, and they realize this ideal deception with the assumption of trusted hardware. In our system, since we make the minimal assumption of an untappable channel between the voter and their hedgehog(s), we cannot realize such a strong DI. Alternatively, we define a weaker DI: If DI chooses to obey, it forwards the coercer's input to the ideal functionality. If DI chooses to deceive, it forwards the coercer's input to the ideal functionality but sends a nullification request at the end of voting phase. Meanwhile, the ideal functionality accepts a special command—"nullify"—and leaks the provisional tally to the simulator $\mathcal{S}$, which captures the features of VoteXX. In this type of weaker ideal deception, the coercer may know that there are people deceiving him, but he cannot attribute such deception to the voter.

Definition C.1. Let π be any protocol and let $\mathcal{F}$ be any ideal functionality. Let $\mathcal{A}$ be the adversary who has the power of corruption and coercion. We say that *π IUC realizes $\mathcal{F}$* if, for every $i \in [n]$ and for every ideal deception strategy DI_i, there exists a real deception strategy DR_i such that, for every PPT adversary $\mathcal{A}$, there exist a simulator $\mathcal{S}$ such that, for any set $\mathsf{DI}_{\mathcal{J}} = \{\mathsf{DI}_i : i \in \mathcal{J}\}$ and any environment $\mathcal{Z}$,

$$\mathsf{EXEC}_{\mathcal{F},\mathsf{DI}_{\mathcal{J}},\mathcal{S},\mathcal{Z}} \approx \mathsf{EXEC}_{\pi,\mathsf{DR}_{\mathcal{J}},\mathcal{A},\mathcal{Z}}. \tag{8}$$

Theorem C.2. *(Universal composition) Let π, ρ be any polynomial-time protocols, and let $\mathcal{F}$ be any ideal functionality. If π IUC realizes $\mathcal{F}$, then ρ^{π} IUC realizes $\rho^{\mathcal{F}}$.*

Following Alwen et al., it is easy to see that our framework remains universally composable with the same type of DI.

The ideal functionality. The voting ideal functionality $\mathcal{F}_{\text{Vote}}^{n,k,t}$ has four phases: preparation, registration, voting, and tally. In the

voting phase, $\mathcal{F}_{\text{Vote}}^{n,k,t}$ receives ballots from the voters and records them. In particular, $\mathcal{F}_{\text{Vote}}^{n,k,t}$ accepts a special type of request: "nullify." Upon receiving a nullify request, the former choice of the voter will not be counted in the final tally. Fig. 5 is a formal description of $\mathcal{F}_{\text{Vote}}^{n,k,t}$.

Extreme coercion. In our UC model, the adversary has the power of extreme coercion. It is modeled as follows. When the adversary $\mathcal{A}$ sends a "extreme coercion" request to a voter, V_i, V_i will hand his state to $\mathcal{A}$ and follow $\mathcal{A}$'s instructions, but V_i can still communicate with his hedgehog(s) H_i secretly, i.e., the communication between V_i and H_i is not controlled by the adversary.

Functionality $\mathcal{F}_{\text{Vote}}^{n,k,t}$

The functionality $\mathcal{F}_{\text{Vote}}^{n,k,t}$ interacts with a set of voters $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, a set of hedgehogs $\mathcal{H} := \{\mathsf{H}_1, \ldots, \mathsf{H}_n\}$, a set of trustees $\mathcal{T} := \{\mathsf{T}_1, \ldots, \mathsf{T}_k\}$, the *Election Authority (EA)*, and the adversary $\mathcal{S}$. Internally it keeps variables status, ballots, τ, and $\mathcal{J}$. Let $\mathcal{P}_{\text{cor}}$ be the set of corrupted parties.
Initially, set status $:= 0$, ballots $:= \tau := \mathcal{J} := \emptyset$.

Preparation:
- Upon receiving (Start, sid) from the trustee $\mathsf{T}_j \in \mathcal{T}$, set $\mathcal{J} := \mathcal{J} \cup \{\mathsf{T}_j\}$, and send a notification (Start, sid, T_j) to the adversary $\mathcal{S}$. (If status $\neq 0$, then ignore the request.)
- Upon receiving (Begin, sid) from the EA, if $|\mathcal{J}| < k$ ignore the request. Otherwise, send a notification (Begin, sid) to the adversary $\mathcal{S}$, and set status $:= 1$. (If status $\neq 0$, then ignore the request.)

Registration:
- Upon receiving (Register, sid) from the voter V_i, send (Register, sid, V_i) to the adversary $\mathcal{S}$. (If status $\neq 1$, then ignore the request.)
- Upon receiving (EndReg, sid) from EA, send (EndReg, sid) to the adversary $\mathcal{S}$ and set status $:= 2$. (If status $\neq 1$, then ignore the request.)

Voting:
- Upon receiving (Vote, sid, x) from a voter $\mathsf{V}_i \in \mathcal{V}$, set ballots$[i] := x$ (x=YES/NO), and send (VoteNotify, sid, V_i) to the adversary $\mathcal{S}$. (If status $\neq 2$, then ignore the request.)
- Upon receiving (EndVote, sid) from EA, compute $\delta \leftarrow$ TallyAlg(ballots) (Cf Fig. 6). Send (PreTally, sid, δ) to the adversary $\mathcal{S}$. Set status $:= 3$. (If status $\neq 2$, then ignore the request.)
- Upon receiving (Nullify, sid) from a voter $\mathsf{V}_i \in \mathcal{V}$ OR V_i's hedgehog H_i, set ballots$[i] :=$ nullify. Send (NullifyNotify, sid) to the adversary $\mathcal{S}$. (If status $\neq 3$, then ignore the request.)

Tally:
- Upon receiving (Tally, sid) from EA, compute $\tau \leftarrow$ TallyAlg(ballots) (Cf Fig. 6). Send (Tally, sid, τ) to the adversary $\mathcal{S}$. (If status $\neq 3$, then ignore the request.)
- Upon receiving (Result, sid) from any party P, if $\tau := \emptyset$, then ignore the request, otherwise return (Result, sid, τ) to the requester.

Figure 5: Functionality $\mathcal{F}_{\text{Vote}}^{n,k,t}$

Connection with the properties. It is easy to see that our UC definition implies the basic properties of a secure voting scheme. First, $\mathcal{F}_{\text{Vote}}^{n,k,t}$ does not leak the ballot of a voter to anyone else, so it implies ballot privacy. Second, as mentioned above, the ideal deception is able to nullify the ballot and the coercer cannot know if the coercion was successful, so our definition implies coercion resistance. Third, $\mathcal{F}_{\text{Vote}}^{n,k,t}$ ensures that the tally procedure is performed correctly, so it implies verifiability.

Tally Algorithm TallyAlg

Input: a table ballots consisting of all the ballots.
Output: tally result σ.
The algorithm performs as follows:
- Set $\mathsf{nr}_{\text{yes}} := 0, \mathsf{nr}_{\text{no}} := 0$.
- For $i := 1$ to n, if ballots$[i]$ = YES then $\mathsf{nr}_{\text{yes}} := \mathsf{nr}_{\text{yes}} + 1$; if ballots$[i]$ = NO then $\mathsf{nr}_{\text{no}} := \mathsf{nr}_{\text{no}} + 1$.
- Return $\sigma := (\mathsf{nr}_{\text{yes}}, \mathsf{nr}_{\text{no}})$

Figure 6: Tally Algorithm TallyAlg

C.5 UC Specification of VoteXX

Before we give a UC proof for VoteXX, we give a UC description of the VoteXX protocol. We assume that the protocol uses "cancel" nullification (Cf. 8.2). We will discuss the security of "flip" variant nullification in Section C.7.

VoteXX protocol $\Pi_{\text{VoteXX}}^{n,k,t}$

Denote the voters as $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, the hedgehogs as $\mathcal{H} := \{\mathsf{H}_1, \ldots, \mathsf{H}_n\}$, the trustees as $\mathcal{T} := \{T_1, \ldots, T_k\}$, and the Election Authority as EA. We assume that EA cannot be corrupted.

Preparation phase:

Upon receiving (Start, sid) from the environment $\mathcal{Z}$, the trustee T_i performs the following:

- Privately choose one random values $\{a_{i,j}\}_{j\in[N-1]}$, reveal $g_0^{a_{i,j}}$, and prove knowledge of $a_{i,j}$ with a Schnorr Σ–Protocol.

Upon receiving (Begin, sid) from the environment $\mathcal{Z}$, the EA performs the following:

- Compute $g_j := \prod_{i=1}^{k} g_0^{a_{i,j}}, j \in [N-1]$.
- Set base $\leftarrow \langle g_0, g_1, g_2, \ldots, g_{N-1}\rangle$ and send (Write, sid, base) to $\mathcal{G}_{\text{BB}}$.

Registration phase:

Upon receiving (Register, sid) from the environment $\mathcal{Z}$, the voter V_i performs the following:

- Send (Read, sid) to $\mathcal{G}_{\text{BB}}$ and get base.
- Generate two N-character passphrases (for YES and NO).
- Parse the passphrase as a sequence of Base64 characters $\langle c_0, c_1, c_2, \ldots, c_N\rangle$ and compute its deterministic commitment using
  base: passCommit $\leftarrow \left\langle g_0^{c_0} \cdot g_1^{c_1} \cdot g_2^{c_2} \cdot \ldots \cdot g_{N-1}^{c_{N-1}} \right\rangle$.
- Send (PassCommit, sid, $\left\langle \mathsf{passCommit}_{\text{yes}}, \mathsf{passCommit}_{\text{no}} \right\rangle$) to the EA.

Upon receiving (PassCommit, sid, $\left\langle \mathsf{passCommit}_{\text{yes}}, \mathsf{passCommit}_{\text{no}} \right\rangle$) from the voter V_i, the EA performs the following:

- Send (Write, sid, $\langle$VoterID, $[\![\mathsf{passCommit}_{\text{yes}}]\!]$, $[\![\mathsf{passCommit}_{\text{no}}]\!]\rangle$) to $\mathcal{G}_{\text{BB}}$, where $[\![\mathsf{passCommit}]\!]$ is an encryption of passCommit under the EA's threshold encryption scheme.

- Prove to the voter client the correctness of the encryptions using the Chaum-Pedersen Σ−Protocol.

Upon receiving (EndReg, sid) from the environment $\mathcal{Z}$, the EA performs the following:

- Take the list of $\left\langle \mathsf{VoterID}, [\![\mathsf{passCommit}_{\mathsf{yes}}]\!], [\![\mathsf{passCommit}_{\mathsf{no}}]\!] \right\rangle$ entries, remove the VoterID component, and verifiably shuffle, threshold-decrypt, and send (Write, sid, $\left\langle \mathsf{passCommit}_{\mathsf{yes}}, \mathsf{passCommit}_{\mathsf{no}} \right\rangle$) to $\mathcal{G}_{\mathsf{BB}}$.
- Send (Reveal, sid) to each trustee T_i.

Upon receiving (Reveal, sid) from EA, trustee T_i sends (Write, sid, $\{a_{i,j}\}_{j\in[N-1]}$) to $\mathcal{G}_{\mathsf{BB}}$.

Upon receiving (EndReg, sid) from the environment $\mathcal{Z}$, the EA sends (StartVote, sid) to each voter.

Voting phase:

Upon receiving (StartVote, sid) from EA, voter V_i performs the following:

- Send (Read, sid) to $\mathcal{G}_{\mathsf{BB}}$ and get $\{a_{i,j}\}_{i\in[k],j\in[N-1]}$.
- Compute $\mathsf{sk} := c_0 + \alpha_1 \cdot c_1 + \alpha_2 \cdot c_2 \ldots$, where $\alpha_i := a_{1,i} + a_{2,i} + \ldots$.
- Send (SecretKey, sid, $\left\langle \mathsf{sk}_{\mathsf{yes}}, \mathsf{sk}_{\mathsf{no}} \right\rangle$) to his hedgehog H_i.

Upon receiving (Vote, sid, x) from the environment $\mathcal{Z}$, voter V_i performs the following:

- If x = YES, generate $\sigma_{\mathsf{yes}} \leftarrow \mathsf{Sign}(\mathsf{nonce})$ and use EA's threshold encryption scheme to compute ballot $\leftarrow \left\langle [\![\mathsf{pk}_{\mathsf{yes}}]\!], [\![\sigma_{\mathsf{yes}}]\!], \pi_{\mathsf{ppk}} \right\rangle$, where each group element of σ is individually encrypted and $\pi_{\mathsf{ppk}} \leftarrow \mathsf{NIZK}_{\mathsf{ballot}}.\mathsf{prove}([\![\mathsf{pk}_{\mathsf{yes}}]\!], [\![\sigma_{\mathsf{yes}}]\!])$ is a proof of plaintext knowledge using the Chaum-Pedersen Σ−Protocol.
- If x = NO, generate $\sigma_{\mathsf{no}} \leftarrow \mathsf{Sign}(\mathsf{nonce})$ and use EA's threshold encryption scheme to compute ballot $\leftarrow \left\langle [\![\mathsf{pk}_{\mathsf{no}}]\!], [\![\sigma_{\mathsf{no}}]\!], \pi_{\mathsf{ppk}} \right\rangle$, where each group element of σ is individually encrypted and $\pi_{\mathsf{ppk}} \leftarrow \mathsf{NIZK}_{\mathsf{ballot}}.\mathsf{prove}([\![\mathsf{pk}_{\mathsf{no}}]\!], [\![\sigma_{\mathsf{no}}]\!])$ is a proof of plaintext knowledge using the Chaum-Pedersen Σ−Protocol.
- Send (Write, sid, ballot) to $\mathcal{G}_{\mathsf{BB}}$.

Upon receiving (EndVote, sid) from the environment $\mathcal{Z}$, the EA performs the following:

- Send (Read, sid) to $\mathcal{G}_{\mathsf{BB}}$ and get the list of $\left\langle [\![\mathsf{pk}]\!], [\![\sigma]\!] \right\rangle$, then threshold-decrypt them: $\langle \mathsf{pk}, \sigma \rangle$. Send (Write, sid, $\langle \mathsf{pk}, \sigma \rangle, \pi$) to $\mathcal{G}_{\mathsf{BB}}$, where $\pi \leftarrow \mathsf{NIZK}_{\mathsf{Dec}}.\mathsf{prove}([\![\mathsf{pk}]\!], [\![\sigma]\!], \mathsf{pk}, \sigma)$ is the decryption NIZK.
- For each ballot, the ballot is marked invalid if σ does not verify under its corresponding pk.
- For each valid signature, determine if it is a YES OR NO key, and count the vote only if it is the only ballot cast that corresponds to that roster entry.
- Use the Roster and set of valid signatures from the provisional tally to reformat the election data into two lists. The first list establishes, in arbitrary order, the set of $\mathsf{pk}_{\mathsf{no}}$ keys from voters who cast valid votes for YES (call it yesVotes). The second list contains $\mathsf{pk}_{\mathsf{yes}}$ from voters who voted NO (call it noVotes).
- Send (Write, sid, ⟨yesVotes, noVotes⟩) to $\mathcal{G}_{\mathsf{BB}}$.
- Send (EndVote, sid) to every voter.

Upon receiving (Nullify, sid, 1) from the voter V_i, the hedgehog H_i does the following:

- Send (Read, sid) to $\mathcal{G}_{\mathsf{BB}}$ and find the key to be nullified in yesVotes OR noVotes. Denote the index of the key as ℓ.
- Prepare a list of encrypted "flags" L marking the ballot it wants to nullify, i.e., the i'th element is $[\![1]\!]$ and the other elements are $[\![0]\!]$.
- Add a proof to this list using the nullification Σ−Protocol. Concisely, the proof statement is: [(this flag is an encryption of 0) OR (this flag is an encryption of 1 and I know $\mathsf{sk}_{\mathsf{no}}$ corresponding to this $\mathsf{pk}_{\mathsf{no}}$)]. Denote the nullification proof as $\pi \leftarrow \mathsf{NIZK}_{\mathsf{nul}}.\mathsf{prove}(\mathsf{yesVotes}/\mathsf{noVotes}, L)$.
- Send (Write, sid, $\langle L, \pi \rangle$) to $\mathcal{G}_{\mathsf{BB}}$.

Upon receiving (Nullify, sid, 0) from the voter V_i, the hedgehog H_i does the following:

- Prepare a list of encrypted "flags" L where all the elements are $[\![0]\!]$.
- Add a proof to this list using the nullification Σ−Protocol. Concisely, the proof statement is: [(this flag is an encryption of 0) OR (this flag is an encryption of 1 and I know $\mathsf{sk}_{\mathsf{no}}$ corresponding to this $\mathsf{pk}_{\mathsf{no}}$)]. Denote the nullification proof as $\pi \leftarrow \mathsf{NIZK}_{\mathsf{nul}}.\mathsf{prove}(\mathsf{yesVotes}/\mathsf{noVotes}, L)$.
- Send (Write, sid, $\langle L, \pi \rangle$) to $\mathcal{G}_{\mathsf{BB}}$.

Tally phase:

Upon receiving (Tally, sid) from the environment $\mathcal{Z}$, EA performs the following:

- Send (Read, sid) to $\mathcal{G}_{\mathsf{BB}}$ and collect all encrypted flags.
- For each $\mathsf{pk}_{\mathsf{no}}$ in yesVotes, take all encrypted flags and compute their OR under encryption using the max-and-match SFE protocol [26].
- Take the list of encrypted ORed flags, sum them under encryption (denote it as c), and verifiably threshold-decrypt the result (denote it as x).
- Subtract this value from the number of YES votes in the provisional tally to produce the final tally for YES votes.
- Repeat the above three steps for each $\mathsf{pk}_{\mathsf{yes}}$ key in noVotes.
- Denote the final tally result as τ. Send (Write, sid, $\langle \tau, \pi \rangle$) to $\mathcal{G}_{\mathsf{BB}}$, where $\pi \leftarrow \mathsf{NIZK}_{\mathsf{Dec}}.\mathsf{prove}(c, x)$ is the decryption proof.

Upon receiving (Result, sid) from the environment $\mathcal{Z}$, the party P returns (Result, sid, τ).

C.6 UC Proof for VoteXX

We have the following theorem.

Theorem C.3. *Assume that* $\mathsf{NIZK}_{\mathsf{ballot}}$, $\mathsf{NIZK}_{\mathsf{nul}}$ *and* $\mathsf{NIZK}_{\mathsf{Dec}}$ *are complete, sound and zero-knowledge; the encryption scheme is IND-CPA secure; and the signature scheme is existentially unforgeable against chosen-message attack. The VoteXX protocol* $\Pi^{n,k,t}_{\mathsf{VoteXX}}$ *IUC realizes* $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$ *against static corruption and mobile extreme coercion in the* $\mathcal{G}_{\mathsf{BB}}$*-hybrid model.*

Proof. To prove the theorem, we need to construct the real deception strategies DR and a simulator $\mathcal{S}$ such that no non-uniform

PPT environment $\mathcal{Z}$ can distinguish (i) the real execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{BB}}}_{\Pi^{n,k,t}_{\mathsf{VoteXX}},\mathsf{DR},\mathcal{A},\mathcal{Z}}$ from the (ii) the ideal execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{BB}}}_{\mathcal{F}^{n,k,t}_{\mathsf{Vote}},\mathsf{DI},\mathcal{S},\mathcal{Z}}$.

Real Deception Strategy. The real deception strategy DR_i internally runs DI_i, forwarding messages to and from the environment $\mathcal{Z}$. DR_i works as follows:

DR_i follows the coercer's instructions. Upon receiving (EndVote, sid) from the EA:

- If DI_i does not send a nullification request to $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, then DR_i sends (Nullify, sid, 0) to the hedgehog H_i via the untappable channel.
- If DI_i sends (Nullify, sid) to $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, then DR_i sends (Nullify, sid, 1) to the hedgehog H_i via the untappable channel.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to and from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ works as follows:

In the preparation phase:

- Upon receiving (Start, sid, T_i) from the functionality $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, $\mathcal{S}$ simulates the trustee T_i following the protocol $\Pi^{n,k,t}_{\mathsf{VoteXX}}$ as if T_i received (Start, sid) from the environment $\mathcal{Z}$.
- Upon receiving (Begin, sid) from the functionality $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, $\mathcal{S}$ simulates the EA following the protocol $\Pi^{n,k,t}_{\mathsf{VoteXX}}$ as if EA received (Begin, sid) from the environment $\mathcal{Z}$.

In the registration phase:

- Upon receiving (Register, sid, V_i) from the functionality $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, the simulator $\mathcal{S}$ simulates V_i following the protocol $\Pi^{n,k,t}_{\mathsf{VoteXX}}$ as if V_i received (Register, sid) from the environment $\mathcal{Z}$.
- Upon receiving (EndReg, sid) from the functionality $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, the simulator $\mathcal{S}$ simulates EA following the protocol $\Pi^{n,k,t}_{\mathsf{VoteXX}}$ as if the EA received (EndReg, sid) from the environment $\mathcal{Z}$.

In the voting phase:

- Upon receiving (VoteNotify, sid, V_i) from the functionality $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, send an encryption of 0 $\langle [\![0]\!], [\![0]\!], \pi_{\mathsf{ppk}} \rangle$ to $\mathcal{G}_{\mathsf{BB}}$.
- If a corrupted party V_i casts a valid ballot for an honest voter on $\mathcal{G}_{\mathsf{BB}}$, $\mathcal{S}$ will abort.
- When a corrupted party V_i casts a ballot on $\mathcal{G}_{\mathsf{BB}}$, decrypt the ballot to get the choice x = YES/NO and send (Vote, sid, x) to $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$ on behalf of V_i in the ideal world.
- Upon receiving (PreTally, sid, δ) from the functionality $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, the simulator $\mathcal{S}$ simulates the pre-tally result by simulating the NIZK for decryption based on δ.
- Upon receiving (NullifyNotify, sid) from the functionality $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, the simulator sends a *dummy nullification request* (a nullification request where the encrypted flags are all $[\![0]\!]$) to $\mathcal{G}_{\mathsf{BB}}$.
- If a corrupted party V_i sends a valid nullification request for an honest voter on $\mathcal{G}_{\mathsf{BB}}$, $\mathcal{S}$ will abort.
- When a corrupted party V_i sends a nullification request on $\mathcal{G}_{\mathsf{BB}}$, decrypt the nullification request. Otherwise, if it is not a dummy nullification request, send (Nullify, sid) to $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$ on behalf of V_i in the ideal world.

In the tally phase:

- Upon receiving (Tally, sid, τ) from the functionality $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$, $\mathcal{S}$ simulates EA doing the following:
  - For each pk in yesVotes and noVotes, take all the encrypted flags, compute its OR under encryption. Sum them under encryption for yesVotes and noVotes, respectively.
  - Simulate the decryption of the summed encrypted flags and the corresponding NIZK proof π based on τ.
  - Send (Write, sid, τ, π) to $\mathcal{G}_{\mathsf{BB}}$.

Indistinguishability.

We prove indistinguishability through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_9$.

Hybrid $\mathcal{H}_0$: This object is the real world execution $\mathsf{EXEC}^{\mathcal{G}_{\mathsf{BB}}}_{\Pi^{n,k,t}_{\mathsf{VoteXX}},\mathsf{DR},\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except the followings. During the pre-tally phase (upon receiving (EndVote, sid) from the environment $\mathcal{Z}$) and tally phase, the EA's decryption proofs are generated by NIZK simulator.

Claim 1: If the decryption NIZK is zero-knowledge with adversary advantage $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{Dec}}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable with distinguishing advantage at most $(2n+2) \cdot \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{Dec}}}(\mathcal{A}, \lambda)$.

Proof: Each ballot has two ciphertexts and there are n voters in total, so there are $2n$ ciphertexts to decrypt in pre-tally. In tally phase, there are 2 ciphertexts to decrypt. Therefore, the overall advantage is at most $(2n+2) \cdot \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{Dec}}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except the followings. During the pre-tally phase (upon receiving (EndVote, sid) from the environment $\mathcal{Z}$) and tally phase, the honest EA members' decryption shares are backward calculated from the pre-tally result and the tally result, respectively.

Claim 2: If the encryption scheme is backward calculatable, then $\mathcal{H}_2$ and $\mathcal{H}_1$ are perfectly indistinguishable.

Proof: The backward calculated decryption shares in $\mathcal{H}_2$ and the decryption shares in $\mathcal{H}_1$ have the same distribution .

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except the followings. During the voting phase, the honest voters' ballot NIZK proofs are generated by the NIZK simulator.

Claim 3: If the ballot NIZK proof is zero-knowledge with adversary advantage $\mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{ballot}}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_3$ and $\mathcal{H}_2$ are indistinguishable with distinguishing advantage at most $n \cdot \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{ballot}}}(\mathcal{A}, \lambda)$.

Proof: There are at most n honest voters, so the overall advantage is at most $n \cdot \mathsf{Adv}^{\mathsf{zk}}_{\mathsf{NIZK}_{\mathsf{ballot}}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_4$: $\mathcal{H}_4$ is the same as $\mathcal{H}_3$ except the followings. During the voting phase, the honest voters' ballots are replaced with $\langle [\![0]\!], [\![0]\!] \rangle$.

Claim 4: If the encryption scheme is IND-CPA secure with advantage $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Enc}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_4$ and $\mathcal{H}_3$ are indistinguishable with distinguishing advantage at most $2n \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Enc}}(\mathcal{A}, \lambda)$.

Proof: There are at most $2n$ ciphertexts in total. Thus, the overall advantage is at most $2n \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\text{Enc}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_5$: $\mathcal{H}_5$ is the same as $\mathcal{H}_4$ except the followings. During the voting phase, honest parties' nullification NIZKs are generated by the NIZK simulator.

Claim 5: If the nullification NIZK proof is zero-knowledge with adversary advantage $\mathsf{Adv}^{\text{zk}}_{\text{NIZK}_{\text{nul}}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_5$ and $\mathcal{H}_4$ are indistinguishable with distinguishing advantage at most $n \cdot \mathsf{Adv}^{\text{zk}}_{\text{NIZK}_{\text{nul}}}(\mathcal{A}, \lambda)$.

Proof: There are at most n honest voters, so the overall advantage is at most $n \cdot \mathsf{Adv}^{\text{zk}}_{\text{NIZK}_{\text{nul}}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_6$: $\mathcal{H}_6$ is the same as $\mathcal{H}_5$ except the followings. During the voting phase, honest parties' nullification requests are replaced with $\langle [\![0]\!], \ldots, [\![0]\!] \rangle$.

Claim 6: If the encryption scheme is IND-CPA secure with advantage $\mathsf{Adv}^{\text{IND-CPA}}_{\text{Enc}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_6$ and $\mathcal{H}_5$ are indistinguishable with distinguishing advantage at most $n \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\text{Enc}}(\mathcal{A}, \lambda)$.

Proof: For each nullification request, there is an encryption of 1 replaced with encryption of 0, and there are at most n honest voters. Thus, the overall advantage is at most $n \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\text{Enc}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_7$: $\mathcal{H}_7$ is the same as $\mathcal{H}_6$ except that, if a corrupted voter generates a valid nullification request for an honest voter, the execution will abort.

Claim 7: If the nullification NIZK is sound with soundness error $\mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{nul}}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_7$ and $\mathcal{H}_6$ are indistinguishable with distinguishing advantage at most $n \cdot \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{nul}}}(\mathcal{A}, \lambda)$.

Proof: There are at most n honest voters, so the probability of aborting is no more than $n \cdot \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{nul}}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_8$: $\mathcal{H}_8$ is the same as $\mathcal{H}_7$ except that, if a corrupted voter generates a valid ballot for an honest voter, the execution will abort.

Claim 8: If the signature scheme is existentially unforgeable against chosen-message attack with adversary advantage $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{Sig}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_8$ and $\mathcal{H}_7$ are indistinguishable with distinguishing advantage at most $\mathsf{Adv}^{\text{EUF-CMA}}_{\text{Sig}}(\mathcal{A}, \lambda)$.

Proof: Same as the previous proof, the probability of aborting is no more than $n \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\text{Sig}}(\mathcal{A}, \lambda)$ by a standard hybrid argument.

Hybrid $\mathcal{H}_9$: This object is the ideal execution $\mathsf{EXEC}^{\mathcal{G}_{\text{BB}}}_{\mathcal{F}^{n,k,t}_{\text{Vote}}, \mathsf{DI}, \mathcal{S}, \mathcal{Z}}$.

Claim 9: If the decryption NIZK is sound with soundness error $\mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{Dec}}}(\mathcal{A}, \lambda)$, the shuffle NIZK is sound with soundness error $\mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{shuffle}}}(\mathcal{A}, \lambda)$ and the max-and-match SFE protocol is robust with adversary advantage $\mathsf{Adv}^{\text{robust}}_{\text{SFE}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_9$ and $\mathcal{H}_8$ are indistinguishable with distinguishing advantage at most $(n+2) \cdot \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{Dec}}}(\mathcal{A}, \lambda) + n \cdot \mathsf{Adv}^{\text{robust}}_{\text{SFE}}(\mathcal{A}, \lambda) + \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{shuffle}}}(\mathcal{A}, \lambda)$.

Proof: To prove Claim 9, we will show that the real tally (pre-tally) and ideal tally (pre-tally) are indistinguishable.

We first show that the real pre-tally and ideal pre-tally are indistinguishable. If decryption correctness of honest voters' ballots holds, the number of yesVotes and noVotes are identical in both real pre-tally and ideal pre-tally. In the ideal execution $\mathsf{EXEC}^{\mathcal{G}_{\text{BB}}}_{\mathcal{F}^{n,k,t}_{\text{Vote}}, \mathsf{DI}, \mathcal{S}, \mathcal{Z}}$, the corrupted parties' ballots are honestly tallied, while honest and coerced parties' pre-tally are simulated by randomly choosing correct number of public keys. They are indistinguishable by the verifiable shuffle in registration phase. Thus, the overall advantage in pre-tally is no more than $n \cdot \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{Dec}}}(\mathcal{A}, \lambda) + \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{shuffle}}}(\mathcal{A}, \lambda)$.

We then show that the real tally and ideal tally are indistinguishable. If the max-and-match SFE protocol is sound and decryption correctness holds, the number of nullified ballots are identical in both real tally and ideal tally. The max-and-match protocol is performed for n times. Thus, the overall advantage in tally is no more than $n \cdot \mathsf{Adv}^{\text{robust}}_{\text{SFE}}(\mathcal{A}, \lambda) + 2 \cdot \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{Dec}}}(\mathcal{A}, \lambda)$.

In summary, the distinguishing advantage of $\mathcal{H}_9$ and $\mathcal{H}_8$ is at most $(n+2) \cdot \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{Dec}}}(\mathcal{A}, \lambda) + n \cdot \mathsf{Adv}^{\text{robust}}_{\text{SFE}}(\mathcal{A}, \lambda) + \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{shuffle}}}(\mathcal{A}, \lambda)$.

Consequently, the real execution $\mathsf{EXEC}^{\mathcal{G}_{\text{BB}}}_{\Pi^{n,k,t}_{\text{VoteXX}}, \mathsf{DR}, \mathcal{A}, \mathcal{Z}}$ and ideal execution $\mathsf{EXEC}^{\mathcal{G}_{\text{BB}}}_{\mathcal{F}^{n,k,t}_{\text{Vote}}, \mathsf{DI}, \mathcal{S}, \mathcal{Z}}$ are indistinguishable with distinguishing advantage no more than

$$(2n+2) \cdot \mathsf{Adv}^{\text{zk}}_{\text{NIZK}_{\text{Dec}}}(\mathcal{A}, \lambda) + n \cdot \mathsf{Adv}^{\text{zk}}_{\text{NIZK}_{\text{ballot}}}(\mathcal{A}, \lambda) +$$
$$3n \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\text{Enc}}(\mathcal{A}, \lambda) + n \cdot \mathsf{Adv}^{\text{zk}}_{\text{NIZK}_{\text{nul}}}(\mathcal{A}, \lambda) + n \cdot \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{nul}}}(\mathcal{A}, \lambda) +$$
$$n \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\text{Sig}}(\mathcal{A}, \lambda) + (n+2) \cdot \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{Dec}}}(\mathcal{A}, \lambda) +$$
$$n \cdot \mathsf{Adv}^{\text{robust}}_{\text{SFE}}(\mathcal{A}, \lambda) + \mathsf{Adv}^{\text{sound}}_{\text{NIZK}_{\text{shuffle}}}(\mathcal{A}, \lambda)$$

This argument concludes the proof. □

□

C.7 Security of "Flip" Variant

In this section, we will illustrate that the above UC proof can be adapted to "flip" nullification. To prove security of "flip" nullification, the ideal functionality $\mathcal{F}^{n,k,t}_{\text{Vote}}$ has interface Flip instead of Nullify. Upon receiving (Flip, sid) from a voter V_i OR V_i's hedgehog, $\mathcal{F}^{n,k,t}_{\text{Vote}}$ flips V_i's vote and sends the notification to $\mathcal{S}$. The ideal deception strategy DI also needs to be modified accordingly: If DI chooses to deceive, it forwards the coercer's input to the ideal functionality and has 50% probability sending a flip request to the ideal functionality.

The construction of the real deception strategy DR_i is straightforward. If DI_i sends a flip request to the ideal functionality, DR_i will follow the protocol to flip the vote. Otherwise, DR_i sends a dummy flip request to BB. The construction of simulator $\mathcal{S}$ is completely the same except that "cancel" request is replaced by "flip" request. Then, through a series of same hybrid worlds, we can prove that "flip" protocol UC realizes $\mathcal{F}^{n,k,t}_{\text{Vote}}$.

D VOTEXX WITH DECOY

In this section, we will show that we can achieve better coercion resistance by adding decoy ballots to VoteXX.

D.1 Intuition

A decoy ballot is a ballot that will not be counted in the final tally, but is indistinguishable from a real ballot in the coercer's view. In

the registration phase, if the voter knows that he will be coerced, he will register two (or more) public keys to the distributed EA and only one of them is real, while the others are decoys. In the voting phase, the voter can use the decoy secret key to submit a ballot that will not be counted in the final tally, and use the real secret key to vote as he wants. However, if the voter cannot hide the real secret key from the coercer, he can still use VoteXX's nullification to nullify the vote. **In conclusion, adding decoy allows a voter to vote as he wants, if he can keep the real secret key from the coercer, and to nullify if he cannot hold the secret.**

To add decoy ballots into VoteXX protocol, we modify the registration phase and the final tally phase while keep the other parts same as VoteXX. We assume that the VoteXX protocol uses "cancel" nullification.

We assume that there is a public roster consisting of commitments of voters' credentials, and each voter holds his credential σ. In the registration phase, if a voter knows that he will be coerced, he (and his hedgehog) sends two (or more) $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}}, \rho \rangle$ tuples to the distributed EA, where $\rho \leftarrow \mathsf{Sign}_\sigma(\mathsf{pk}_{\mathsf{yes}} || \mathsf{pk}_{\mathsf{no}})$. At the end of the registration phase, the distributed EA will a perform *secure multi-party computation (MPC)* to generate the table of public keys, which is of the form $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}}, i \rangle$, where $i \in \{[\![0]\!], [\![1]\!]\}$ is called a "decoy flag". If $i = [\![1]\!]$, it means that the corresponding public key is decoy. For the public keys signed by the same credential, the distributed EA uses a public function to determine which public key is real. The secure MPC ensures that none of the EA trustees know which of the public keys are decoys.

In the final tally phase, for each key in yesVotes OR noVotes, the EA takes all the encrypted flags and the corresponding "decoy flags" to compute them OR under encryption using the max-and-match SFE protocol [26]. In this way, if the ballot is a decoy ballot, it will be nullified automatically since the "decoy flag" is an encryption of 1; if the ballot is a real ballot, it can be nullified the same way as in VoteXX.

D.2 Security Definition

We analyze security of VoteXX with decoy under the UC framework [7]. Comparing with VoteXX, the only difference is the ideal deception strategy.

Ideal deception. VoteXX with decoy realizes a stronger DI, which performs as follows: when DI receives an input x from the simulator (ideal coercer) $\mathcal{S}$, it maps x to x' and sends (Vote, sid, x') to the ideal functionality $\mathcal{F}$. (x' can be equal OR not equal to x, representing obeying and deceiving, respectively.)

D.3 UC Specification of VoteXX with Decoy

VoteXX with decoy protocol $\Pi^{n,k,t}_{\mathsf{VoteXX-decoy}}$

Denote the voters as $\mathcal{V} := \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$, the hedgehogs as $\mathcal{H} := \{\mathsf{H}_1, \ldots, \mathsf{H}_n\}$, the trustees as $\mathcal{T} := \{T_1, \ldots, T_k\}$, and the *Election Authority* as EA. We assume that EA cannot be corrupted. At the beginning of the protocol, each voter holds his credential σ and $\mathcal{G}_{\mathsf{BB}}$ contains commitments of σ.

Preparation phase:

Upon receiving (Begin, sid) from the environment $\mathcal{Z}$, the EA performs the initialization procedure of the secure multi-party computation.

Registration phase:

Upon receiving (Register, sid) from the environment $\mathcal{Z}$, the voter V_i performs the following:

- Send $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}}, \rho \rangle$ to the EA, where $\rho \leftarrow \mathsf{Sign}_\sigma(\mathsf{pk}_{\mathsf{yes}} || \mathsf{pk}_{\mathsf{no}})$, and hold the corresponding $\langle \mathsf{sk}_{\mathsf{yes}}, \mathsf{sk}_{\mathsf{no}} \rangle$.
- (If V_i will be coerced, he sends $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}}, \rho \rangle$ and $\langle \mathsf{pk}'_{\mathsf{yes}}, \mathsf{pk}'_{\mathsf{no}}, \rho' \rangle$ to the EA, where $\rho \leftarrow \mathsf{Sign}_\sigma(\mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}})$ and $\rho' \leftarrow \mathsf{Sign}_\sigma(\mathsf{pk}'_{\mathsf{yes}} || \mathsf{pk}'_{\mathsf{no}})$. He holds the corresponding $\langle \mathsf{sk}_{\mathsf{yes}}, \mathsf{sk}_{\mathsf{no}} \rangle$ and $\left\langle \mathsf{sk}'_{\mathsf{yes}}, \mathsf{sk}'_{\mathsf{no}} \right\rangle$. Let $\langle \mathsf{sk}_{\mathsf{yes}}, \mathsf{sk}_{\mathsf{no}} \rangle$ be the real key and $\left\langle \mathsf{sk}'_{\mathsf{yes}}, \mathsf{sk}'_{\mathsf{no}} \right\rangle$ be the decoy key.)

Upon receiving (EndReg, sid) from the environment $\mathcal{Z}$, the EA performs the following:

- Perform secure multi-party computation to generate a list of the form $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}}, i \rangle$, where $i \in \{[\![0]\!], [\![1]\!]\}$. For the public keys signed by the same credential, the EA uses a public function h to determine which public key is real. The real keys have $i = [\![0]\!]$ while the decoy keys have $i = [\![1]\!]$.
- Send (Write, sid, $\{\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}}, i \rangle\}$) to $\mathcal{G}_{\mathsf{BB}}$.

Voting phase is completely the same as $\Pi^{n,k,t}_{\mathsf{VoteXX}}$.

Tally phase:

Upon receiving (Tally, sid) from the environment $\mathcal{Z}$, EA performs the following:

- Send (Read, sid) to $\mathcal{G}_{\mathsf{BB}}$ and collect all encrypted flags and decoy flags.
- For each $\mathsf{pk}_{\mathsf{no}}$ in yesVotes, take all encrypted flags **along with the decoy flag** and compute their OR under encryption using the max-and-match SFE protocol [26].
- Take the list of encrypted ORed flags, sum them under encryption, and verifiably threshold-decrypt the result.
- Subtract this value from the number of YES votes in the provisional tally to produce the final tally for YES votes.
- Repeat the above three steps for each $\mathsf{pk}_{\mathsf{yes}}$ key in noVotes.
- Denote the final tally result as τ. Send (Write, sid, τ, π) to $\mathcal{G}_{\mathsf{BB}}$, where π is the NIZK for SFE and decryption.

Upon receiving (Result, sid) from the environment $\mathcal{Z}$, the party P returns (Result, sid, τ).

D.4 UC Proof for VoteXX with Decoy

By constructing a similar simulator, we have the following theorem. Since we need to assume the voter knows he will be coerced in the registration phase, the VoteXX with decoy protocol is secure against static active coercion.

Theorem D.1. *Assume that the NIZKs are complete, sound and zero-knowledge; the encryption scheme is IND-CPA secure; and the signature scheme is secure against existential forgery. The "VoteXX with decoy" protocol $\Pi^{n,k,t}_{\mathsf{VoteXX-decoy}}$ IUC realizes $\mathcal{F}^{n,k,t}_{\mathsf{Vote}}$ against static corruption and static active coercion in the $\mathcal{G}_{\mathsf{BB}}$-hybrid model.*

Proof sketch. To prove the theorem, we need to construct the real deception strategies DR and a simulator $\mathcal{S}$ such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish (i) the real execution $\mathsf{EXEC}^{\mathcal{G}_{\mathrm{BB}}}_{\Pi^{n,k,t}_{\text{VoteXX-decoy}},\mathsf{DR},\mathcal{A},\mathcal{Z}}$ from the (ii) the ideal execution $\mathsf{EXEC}^{\mathcal{G}_{\mathrm{BB}}}_{\mathcal{F}^{n,k,t}_{\mathrm{Vote}},\mathsf{DI},\mathcal{S},\mathcal{Z}}$.

Real Deception Strategy. The real deception strategy DR_i internally runs DI_i, forwarding messages to and from the outside. DR_i works as follows:

In the registration phase, upon receiving (Register, sid) from the environment $\mathcal{Z}$, V_i sends $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}}, \rho\rangle$ and $\langle \mathsf{pk}'_{\mathsf{yes}}, \mathsf{pk}'_{\mathsf{no}}, \rho'\rangle$ to the EA, where $\rho \leftarrow \mathsf{Sign}_\sigma(\mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}})$ and $\rho' \leftarrow \mathsf{Sign}_\sigma(\mathsf{pk}'_{\mathsf{yes}}||\mathsf{pk}'_{\mathsf{no}})$, and holds the corresponding $\left\langle \mathsf{sk}_{\mathsf{yes}}, \mathsf{sk}_{\mathsf{no}}\right\rangle$ (real key) and $\left\langle \mathsf{sk}'_{\mathsf{yes}}, \mathsf{sk}'_{\mathsf{no}}\right\rangle$ (decoy key).

In the voting phase:

- When coerced, DR_i provides the decoy key $\mathsf{sk}'_{\mathsf{yes}}, \mathsf{sk}'_{\mathsf{no}}$ to the coercer and simulates the transcript of using $\mathsf{sk}'_{\mathsf{yes}}, \mathsf{sk}'_{\mathsf{no}}$ to register. DR_i follows the coercer's instructions.
- If DI_i sends (Vote, sid, x') to the ideal functionality, DR_i performs as if V_i receives (Vote, sid, x') from the environment $\mathcal{Z}$.

Simulator. The simulator $\mathcal{S}$ is almost the same as the simulator for $\Pi^{n,k,t}_{\mathrm{VoteXX}}$ since we only modified the registration phase and tally phase.

In the registration phase, the simulator $\mathcal{S}$ simulates the parties following the protocol:

- Upon receiving (Register, sid, V_i) from the functionality $\mathcal{F}^{n,k,t}_{\mathrm{Vote}}$, the simulator $\mathcal{S}$ simulates V_i following the protocol $\Pi^{n,k,t}_{\text{VoteXX-decoy}}$ as if V_i received (Register, sid) from the environment $\mathcal{Z}$.
- Upon receiving (EndReg, sid) from the functionality $\mathcal{F}^{n,k,t}_{\mathrm{Vote}}$, the simulator $\mathcal{S}$ simulates EA following the protocol $\Pi^{n,k,t}_{\text{VoteXX-decoy}}$ as if the EA received (EndReg, sid) from the environment $\mathcal{Z}$.

In the tally phase, the simulator $\mathcal{S}$ simulates the EA to perform the tally procedure, but uses the NIZK simulator to generate the NIZK proof:

- Upon receiving (Tally, sid, τ) from the functionality $\mathcal{F}^{n,k,t}_{\mathrm{Vote}}$, $\mathcal{S}$ simulates EA doing the following:
  - For each pk in yesVotes and noVotes, take all the encrypted flags **along with the decoy flag**, compute its OR under encryption. Sum them under encryption for yesVotes and noVotes, respectively.
  - Simulate the decryption of the summed encrypted flags and the corresponding NIZK π based on τ.
  - Send (Write, sid, τ, π) to $\mathcal{G}_{\mathrm{BB}}$.

Same as the proof of **Theorem A.2**, through a series of hybrid worlds, the real world and ideal world are indistinguishable.