VoteXX: A Solution to Improper Influence in Voter-Verifiable Elections

Abstract. We solve a long-standing challenge to the integrity of votes cast without the supervision of a voting booth: "*improper influence*," which typically refers to any combination of vote buying and voter coercion. 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, irrevocable, and forever unattributable to the voter. In particular, our approach enhances security of online, remote, public-sector elections, for which there is a growing need and the threat of improper influence is most acute.

We introduce the new approach, give detailed cryptographic protocols, show how it can be applied to several voting settings, and describe our implementation. The protocols compose a full voting system, which we call *VoteXX*, including registration, voting, nullification, and tallying—using an anonymous communication system for registration, vote casting, and other communication in the system. We demonstrate how the technique can be applied to known systems, including where ballots can be mailed to voters and voters use codes on the ballot to cast their votes online. In comparison with previous proposals, our system makes fewer assumptions and protects against a strong adversary who learns all of the voter's keys.

In VoteXX, each voter has two public-private key pairs. Without revealing their private keys, each voter registers their public keys with the election authority. Each voter may share their keys with one or more hedgehogs. During nullification, the voter, or one or more of their hedgehogs, can interact through the anonymous communication system to nullify a vote by proving knowledge of one of the voter's private keys via a zero-knowledge proof without revealing the private key. We describe a fully decentralizable implementation of VoteXX, including its public bulletin board, which could be implemented on a blockchain.

Keywords: anonymous communication system · 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.

1 Introduction

For over 150 years, the voting booth helped prevent voters from being bribed and coerced. 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, technology facilitates vote buying and voter coercion with

electronic payments, live video streaming from voter phones, and various types of online threats.

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 the 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 knowledge of the voter's secret key using a zero-knowledge proof without revealing the secret key. 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.

Our approach differs from previous approaches (see Sect. 2)—e.g., revoting, fake credentials, and decoy ballots—by leveraging the realistic assumption of an untappable channel between the voter and their hedgehog(s). For instance, our system does not have to make any of the following strong assumptions, which can be readily violated by realistic adversaries: an untappable registration channel, a final time when the voter can vote securely, or that voters are willing to help discourage vote buying by selling decoy ballots. We protect against what we believe to be the strongest possible adversarial model (apart from coercers blocking registration or voting), in which adversaries can learn all voter secrets and observe all voter interactions with the system (excluding interactions with the hedgehogs).

Election system designers face many other challenges: outcome integrity, ballot privacy, usability, accessibility, voter authentication, voting method (e.g., plurality, range voting), definitiveness, turnout, public verifiability, scalability, compatibility with standard data formats, unstoppability, cost, and protection against malware.

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 the voter's device (e.g., phone) might undetectably modify votes and spy on voters. (3) Determined adversaries might try to launch an online attack, including causing outages. Of these challenges, the most elusive has been mitigating improper influence.

In this paper, we present a new solution to improper influence, which many people consider to be the most acute challenge for online voting. Along the way, the decentralized architecture and design of our VoteXX system enhances approaches to other challenges, including unstoppability and protection against malware. In addition, we maintain state-of-the-art protection for outcome integrity, public verifiability, ballot privacy, and scalability.

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 it 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 imple-

mentation of VoteXX, which uses an *anonymous communication system (ACS)* for registration, vote casting, and other communication. In addition, while other systems complicate registration and vote casting, our approach allows simple registration and vote casting by keeping nullification separate.

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. 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 voter can prove to any coercer that the voter cast a counted ballot according to the coercer's instructions. Smyth [30] surveys four definitions of coercion resistance and finds that "coercion resistance has not been adequately formalized." Three of the definitions are too weak, and the general definition by Küsters [22] is complex and too strong. Similarly, there remains some debate on the definition of receipt freeness [11].

Table 1 compares our solution to previous proposed mechanisms. Previous work often makes strong assumptions: the voter knows an honest *Election Authority (EA)* official [8]; the voter needs a special device to evade coercion [2,3,8,19]; the voter needs to perform mental arithmetic to evade coercion [34]; the voter needs to generate a fake password to evade coercion [7,12]; the voter must complete registration before being coerced [19]; the election will not close before the voter can cast a ballot after coercion [24,31,33]; and the probability of successful coercion is lowered by flooding voters with decoy ballots [5]. VoteXX makes none of these assumptions.

We do assume the voter can use an *untappable channel*, as all coercion-resistant system must—if an adversary can always influence the voter, they are indistinguishable from the voter [16]. 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. Since an adversary could always prevent a voter from voting, VoteXX achieves an optimal solution. 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 is unable to re-vote (due to coercion at the end of the election), nullification is a failsafe.

Table 1. Strategies for resisting improper influence in *end-to-end (E2E)* verifiable elections. All properties are with respect to coercion-resistance. Properties are fully present (●), partially present (◐), or not present (○). Full explanation in full version of paper. Decoy ballots act indirectly against influence and receive ◐.

Type	*Example*	Tolerates dishonest EA	No special device required	Low cognitive burden	Tolerates coercion at any time	Can fully override coercion	Inexpensive
Baseline (coercible)	Helios (2008) [1]	○	○	○	○	○	●
Fake credentials	JCJ (2005) [19]	◐	○	●	○	●	●
Masked ballots	WeBu09 (2009) [34]	●	●	○	○	◐	●
Panic passwords	Selections (2011) [7]	●	●	◐	●	●	●
Digital decoy ballots	RS-Voting (2012) [5]	●	◐	◐	◐	◐	○
Re-voting	VoteAgain (2020) [24]	●	●	●	○	●	●
Hedgehogs	This work (2022)	●	●	●	●	◐	●

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 hedgehogs, can interact with the ACS to nullify a vote by proving knowledge of one of the voter's private keys via a *zero-knowledge proof (ZKP)* without revealing the private key. 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 voter registration.

We assume a secure ACS that protects against traffic analysis. Examples include TOR with hidden services [32], I2P [17], xx network [36], and Oxen [26].

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 Sect. 2.

3.2 Problem Specification

Our main requirement is a coercion-resistant remote voting system that achieves a level of security as close as possible to a precinct based in-person voter verifiable secret paper ballot system. The system must maximize the ability to prevent or remediate serious failures by eliminating undetectable attacks, prevent scalable "wholesale" attacks, and make "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 it was coerced. 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. Undetected changes in the software or hardware must not, for all time, yield undetectable changes in the result. This property is similar to *software independence* but with the caveat that before or after the election a version of the software exists without the undetected change. 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 be resistant to denial of service attacks, and no single entity should be able to prevent completion of the election.

In practice, the system must also meet the requirements explained by Park *et al.* [27]. Specifically, the system must have *voter verifiable ballots*, *contestability*, and *auditing*, in addition to the specific requirements mentioned above.

3.3 System Architecture

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 three entities, each under the control of a set of trustees: a *Registration Authority (RA)*, a *Voting Authority (VA)*, and a *Tallying Authority (TA)*. 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 that the operations performed by the EA and via the ACS are correct.

4 Protocols

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

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 [10], which can be augmented with *distributed key generation (DKG)* and threshold decryption (for m out of n key holders [28]). We use standard $\Sigma-$Protocols to prove knowledge of discrete logarithms (Schnorr [29]), knowledge of Diffie-Hellman tuples (Chaum-Pedersen [6]), which also corresponds to ElGamal re-randomizations and decryptions, and knowledge of representations (Okamoto [25]). We also use techniques to allow the trustees to compute jointly and verifiably (*i.e.,* produce $\Sigma-$Protocol proofs), and to compute 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 [18]).

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. 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 network and communication metadata.

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 will 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.* [7]). For simplicity, the protocol boxes do not explain that, for nullification, we use exponential ElGamal [10] 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 an $\Sigma-$Protocol for each flag.

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 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 $\Sigma-\mathsf{Protocol}$. Then $g_1 = g_0^{(a_1 + a_2' + a_3'' + \ldots)}$.
3. This process is repeated, with new random a_i values, to complete a set of N generators: $\mathsf{base} \leftarrow \langle g_0, g_1, g_2, \ldots, g_{N-1} \rangle$. The same base is used for all voters in a registration period.
4. Call the set of all a values (split across the trustees): $\mathsf{trapdoor}$.

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.
4. The EA officer issues a challenge like: "Reveal Character 4." The voter responds "F." The EA client computes $\mathsf{disclosedChar} \leftarrow (\mathsf{passCommit}/g_4^{\mathtt{F}})$. The voting client proves knowledge of a representation of $\mathsf{disclosedChar}$ using a $\Sigma-\mathsf{Protocol}$. 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, where $[\![\mathsf{passCommit}]\!]$ is an encryption of $\mathsf{passCommit}$ under the EA's threshold encryption scheme. Finally the EA client proves to the voter client the correctness of the encryptions using the Chaum-Pedersen $\Sigma-\mathsf{Protocol}$.

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 $\mathsf{VoterID}$ 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 $\mathsf{trapdoor}$.
3. Each voter uses $\mathsf{trapdoor}$ to reformat their two $\mathsf{passCommit}$ 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 (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 nonce 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 $\mathsf{ballot} \leftarrow \langle [\![pk_{\mathsf{yes}}]\!], [\![\sigma_{\mathsf{yes}}]\!], \pi_{\mathsf{ppk}} \rangle$, where each group element of σ is individually encrypted and π_{ppk} is a proof of plaintext knowledge using the Chaum-Pedersen $\Sigma-\mathsf{Protocol}$.
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.

The $\Sigma-\mathsf{Protocol}$ 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 $(\mathsf{flag} = [\![0]\!])$. 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 $\langle g, c_1, y, c_2 \hat{m}^{-1} \rangle$ is a DDH tuple, which can be done with the Chaum-Pedersen $\Sigma-\mathsf{Protocol}$. Call this subproof A. In $\Sigma-\mathsf{Protocol}$ format, its transcript is $\langle a_A, e_A, z_A \rangle$.

In Case 2, the hedgehog proves a conjunctive statement: $(\mathsf{flag} = [\![1]\!])$ and it knows sk, which corresponds to pk for the associated voter's public key. Call the subproof that $(\mathsf{flag} = [\![1]\!])$ B. It is implemented the same as 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 $\Sigma-\mathsf{Protocol}$ due to Schnorr: $\langle a_C, e_C, z_C \rangle$. To summarize, the hedgehog proves: $\Pi :=$ [A `OR` (B `AND` C)].

It is well known that $\Sigma-\mathsf{Protocols}$ can be stacked through conjunction and disjunction [9,14]. Further, the resulting proof can be made non-interactive (typically in the random oracle model with the Fiat-Shamir heuristic [13], in its strong form [4], but other heuristics exist [15]). 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 $\langle a_C, e_B, z_C \rangle$.

Once a set of flags (each real or false with its own proof Π) is computed and submitted by a hedgehog, 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—it is quadratic work (number of hedgehogs times number of voters) and subject to "board flooding" attacks [21]. It must process the nullifications as they arrive ("concurrent authorization" [12]). Doing so is possible. When a new set of flags arrives, each proof is checked and the xor between the submitted flag and the accumulation of previous flags is computed (both are parallelizable for each

Provisional Tally. After the voting period is over, the EA produces a verifiable provisional tally.

1. The EA takes the list of $\langle [\![\mathsf{pk}]\!], [\![\sigma]\!] \rangle$, then verifiably shuffles and 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. Assume the following strategy: the voter wants to vote YES and reveals $\mathsf{sk}_{\mathsf{yes}}$ to the hedgehog, along with $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}} \rangle$. 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 $\Sigma-$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 $\mathsf{pk}_{\mathsf{no}}$)].

Final Tally. After the nullification period is over, 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 xor under encryption using the mix and match SFE protocol [18]. It repeats this process for the remaining $\mathsf{pk}_{\mathsf{no}}$ keys.
2. The EA takes the list of encrypted xored 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).

flag). Thus, when nullification closes, the only remaining task is to threshold decrypt the accumulation of flags, which is linear in number of votes.

5 Design, Client Interfaces, and Implementation

Design elements. The design of VoteXX differs from that of 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, pre-existing 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.

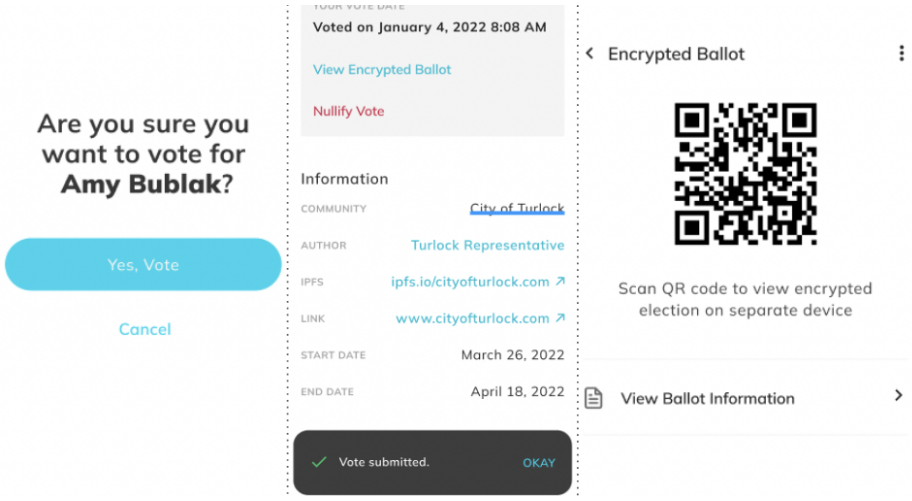

Fig. 1. Voter interface. The voting client generates ballots when the voter enters the proper YES or NO passphrase (left), optionally both phrases. Any client can nullify (middle), and users can scan their ballot QR code (right) to any other voting client instead of using their passphrase.

Client interfaces. The voter clients operate like a calculator, without state or persistent storage. The voter can enter their YES or NO passphrase on another device at any time to regenerate their ballot, and the client will verify that the ballot is properly posted to the BB. Any voter can be a hedgehog for themselves or other voters.

The voter and hedgehog clients are integrated into the same mobile phone app as shown in Fig. 1. Voters and hedgehogs can also send and receive ballot secrets directly between each other using their ACS identities.

The EA client starts 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.

Implementation. We implemented the EA and auditor binaries in Java and Golang as ACS clients that interact with the BB. Our implementation uses the MODP-4096 [20] DH group for tallying and nullification. We use the BouncyCastle [23] library and native Golang cryptography library for all cryptographic operations.

The election parameters file establishes a public 256-bit ACS identity for all observers. All clients use a file-based input/output for each operation, where each file corresponds to a specific *uniform resource locator (URL)* with the file content in a standardized JSON format. Each BB instance is a listener to the ACS identity, storing the data to disk using the same URL structure. Anyone can run a BB instance that provides a REST-like API to access the files through the ACS, and they can host access to these files using separate HTTPS or SFTP servers.

6 Discussion

We now discuss our major design decisions, nullification options (cancel, cancel-toggle, and flip), analysis, extensions, and open problems.

6.1 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 provides key functionality for *malware resistance*.

Nullification and hedgehogs. Nullification allows the voter to share a passphrase anytime after they conceive of it. In-person registrations 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 [21] 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.

In-Person registration. Our registration design roots trust into passphrases known to the voter and written on physical paper attached to 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.

Undetectable wholesale attacks are stacked asymmetrically against an adversary: they must deploy malicious software across all devices controlled by checking with a passphrase. The deployment must go undetected forever—or, if they do not care about loss of confidence from a provably improper election outcome, at least until the election completes. Therefore, the adversary must assume a level of sophistication against that of the best nation state actors if the adversary is to remain undetected for a useful period of time. These decisions allow VoteXX to prevent undetectable wholesale attacks at scale and provide detection and mitigation against retail attacks.

Passphrases also raise potential challenges: they are a target for malware, and the system must be prepared to deal with voters who forget theirs.

6.2 Cancel, Cancel-Toggle, or Flip

Our design supports a variety of options for implementing the semantics of nullification, including what we call "cancel," "cancel-toggle," or "flip." Consider a vote that might have been nullified by one or more entities. Assume that this vote selects from one of k ballot choices numbered $0, 1, \ldots, k-1$ (see multi-candidates in Sect. 6.5). With *cancel*, the vote is cancelled if and only if at least one entity nullified it (and this can be generalized to at least t entities for some threshold t). With *cancel-toggle*, the vote is cancelled if and only if an odd number of entities nullified it. With *flip*, the vote becomes $x + y \mod k$, where x is the ballot choice of the vote, and y is the number of times the vote was nullified.

Each of these options can be implemented using different algebraic operations during Step 1 of the third phase (Final Tally) of Protocol 3: respectively, `AND` for cancel (realized with a homomorphic addition of the encrypted flags for each ballot followed by a plaintext equality test with $[\![0]\!]$), `XOR` for cancel-toggle (as described, using mix and match [18]), and `ADDITION` modulo k for flip (realized with mix and match. The final summation in Step 2 is replaced with a verifiable shuffle and threshold decryption of the flag set for each key).

Intuitively, cancel gives the voter the ability to cancel the vote, whereas flip gives the voter the ability to randomize the vote. Cancel-toggle has the undesirable property that the voter cannot guarantee to cancel or randomize the vote. For example, in an election selecting Alice or Bob, suppose coercer Charlie forces voter Victor to vote for Bob in his presence. Secretly, Charlie randomly chooses whether to nullify. Regardless of whether Victor nullifies, with cancel-toggle the result will be the same: there is 50% chance that the vote is nullified, and there is a 50% chance that the vote will remain for Bob. We recommend not using cancel-toggle.

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.

6.3 Security Note

A limitation of our work is that we do not have formal statements and proofs of the privacy and security properties of VoteXX, which we consider beyond the scope of this paper. These properties stem, in part, from VoteXX's use of an ACS and ZKPs. For example, the EA does not know which public key corresponds to which voter. No party can learn the passphrase from the values disclosed privately and publicly by the voter. We are working on formal statements of the privacy and security properties of VoteXX and a UC proof.

6.4 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 $\Sigma-$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 finalization, generally taking $\Theta(VT \log V)$. Each vote has a constant amount of signatures, encryptions, and $\Sigma-$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 Sect. 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 $\Sigma-$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 $\Sigma-$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 $\Sigma-$Protocols) for the trustees and auditors, with same order proof size.

6.5 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 scalability of the nullification protocol is linear 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 [37].

Malware. 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 [35].

6.6 Open Problems and Future work

Open problems include refining a formal definition of coercion resistance.

It might be possible to devise a way for voters to validate correct construction of a voter's ballot posted in the public record, without revealing how the voter voted. For example, it may be possible to do so by revealing part of the voter's passphrase, enabling a kind of ZKP.

It would be interesting to explore how the number of nullifications might provide a measure of coercion. 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.

The next steps for VoteXX are a formal security proof, formal protocol verification, and a formally verified implementation of key system elements. We also plan to conduct a pilot election and user study to assess the overall usability. 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.

7 Conclusion

We have presented a new, practical, and flexible way to remove improper influence from election systems, 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 rerunning 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 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.

References

1. Adida, B.: Helios: Web-Based open-audit voting. In: USENIX Security Symposium. pp. 335–348 (2008)
2. Araujo, R., Foulle, S., Traoré, J.: A practical and secure coercion-resistant scheme for Internet voting. Toward Trustworthy Elections **LNCS 6000** (2010)
3. Araujo, R., Rajeb, N.B., Robbana, R., Traoré, J., Yousfi, S.: Towards practical and secure coercion-resistant electronic elections. In: CANS (2010)
4. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In: ASIACRYPT (2012)
5. Chaum, D.: Random-Sample Voting (2012), online
6. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO (1992)
7. Clark, J., Hengartner, U.: Selections: Internet voting with over-the-shoulder coercion-resistance. In: Financial Cryptography (2011)
8. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: IEEE Symposium on Security and Privacy. pp. 354–368 (2008)
9. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: CRYPTO (1994)
10. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multiauthority election scheme. In: EUROCRYPT (1997)
11. Delaune, S., Kremer, S., Ryan, M.: Coercion-Resistance and receipt-freeness in electronic voting. In: 19th IEEE Computer Security Foundations Workshop (CSFW'06). pp. 12+. IEEE (2006)
12. Essex, A., Clark, J., Hengartner, U.: Cobra: Toward concurrent ballot authorization for Internet voting. In: EVT/WOTE (2012)
13. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986)
14. Fiat, A., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: ACM STOC (1990)
15. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols. Springer (2010)

16. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: EUROCRYPT (2000)
17. I2P: Invisible Internet project. https://www.geti2p.net, accessed on 01-05-2022
18. Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: ASIACRYPT (2000)
19. Juels, A., Catalano, D., Jacobsson, M.: Coercion-Resistant electronic elections. In: ACM WPES (2005)
20. Kivinen, T., Kojo, M.: More modular exponential (modp) Diffie-Hellman groups for Internet key exchange (IKE). RFC 3526, IETF (2003)
21. Koenig, R., Haenni, R., Fischli, S.: Preventing board flooding attacks in coercion-resistant electronic voting schemes. In: SEC (2011)
22. Küsters, R., Truderung, T., Vogt, A.: Accountability: Definition and relationship to verifiability. In: ACM CCS (2010)
23. Legion of the Bouncy Castle: Bouncy Castle crypto APIs, https://www.bouncycastle.org/java.html, accessed: 2022-05-05
24. Lueks, W., Querejeta-Azurmendi, I., Troncoso, C.: Voteagain: A scalable coercion-resistant voting system. In: USENIX Security (2020)
25. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: CRYPTO (1992)
26. Oxen: Privacy made simple. https://www.oxen.io, accessed on 01-05-2022
27. Park, S., Specter, M., Narula, N., Rivest, R.L.: Going from bad to worse: From Internet voting to blockchain voting (2020), online
28. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: EUROCRYPT (1991)
29. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptography **4**, 161–174 (1991)
30. Smyth, B.: Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822 (2019)
31. Spycher, O., Haenni, R., Dubuis, E.: Coercion-Resistant hybrid voting systems. In: EVOTE (2010)
32. TOR: The TOR project. https://www.torproject.org, accessed on 01-05-2022
33. Volkamer, M., Grimm, R.: Multiple casts in online voting: Analyzing chances. In: EVOTE (2006)
34. Wen, R., Buckland, R.: Masked ballot voting for receipt-free online elections. In: VOTE-ID (2009)
35. Wikipedia: Identity verification service, https://en.wikipedia.org/wiki/Identity_verification_service, accessed: 2022-05-07
36. xx.network: A quantum leap in privacy. https://www.xx.network, accessed on 01-05-2022
37. Zagórski, F., Carback, R., Chaum, D., Clark, J., Essex, A., Vora, P.L.: Remotegrity: Design and use of an end-to-end verifiable remote voting system. In: ACNS (2013)