Revisiting Silent Coercion

David Chaum[1], Richard T. Carback[1], Jeremy Clark[2(✉)], Chao Liu[3], Mahdi Nejadgholi[2], Bart Preneel[4], Alan T. Sherman[3], Mario Yaksetig[1], Zeyuan Yin[6], Filip Zagórski[5], and Bingsheng Zhang[6]

[1] xx.network, Los Angeles, USA
[2] Concordia University, Montreal, Canada
j.clark@concordia.ca
[3] Cyber Defense Lab, University of Maryland, Baltimore County (UMBC), Baltimore, USA
[4] COSIC, KU Leuven and imec, Leuven, Belgium
[5] Department of Computer Science, Wroclaw University of Technology, Wroclaw, Poland
[6] Zhejiang University, Hangzhou, China

Abstract. We revisit "silent coercion" where an adversary gains access to a voter's credential without the voter's knowledge in an E2E verifiable, coercion-resistant Internet voting system. We argue that in this setting, casting an intended vote is impossible since the cryptographic backend can no longer distinguish the voter and adversary. However, we affirm that the voter can still act to nullify adversarial ballots, which is preferable to inaction. We provide a new instantiation of nullification using zero-knowledge proofs and multiparty computation, which improves on the efficiency of the current state-of-the-art. We also demonstrate an example voting system—VoteXX—that uses nullification. Our nullification protocol can complement new and existing techniques for coercion resistance (which all require voters to hide cryptographic keys from the coercer), providing a failsafe option for voters whose keys leak.

1 Our Contributions in Context

A well-studied application of cryptography is to voting systems, including those used in real-world governmental elections [5]. Among other things, cryptography can help provide election integrity (the final tally correctly reflects all ballots exactly as cast), while maintaining secret ballots—referred to as *end-to-end (E2E)* verifiable voting. These schemes generally offer unconditional integrity and rely a set of trustees, which constitute an EA where ballot secrecy is preserved if m-out-of-n trustees are honest (for a configurable m and n). We study undue influence and credential loss in the setting of remote or Internet voting.

An abstract of this paper appeared previously [8]. A technical report with expanded details and security proofs is also available [9].

D. Duenas-Cid et al. (Eds.): E-Vote-ID 2025, LNCS 16028, pp. 38–54, 2026.
https://doi.org/10.1007/978-3-032-05036-6_3

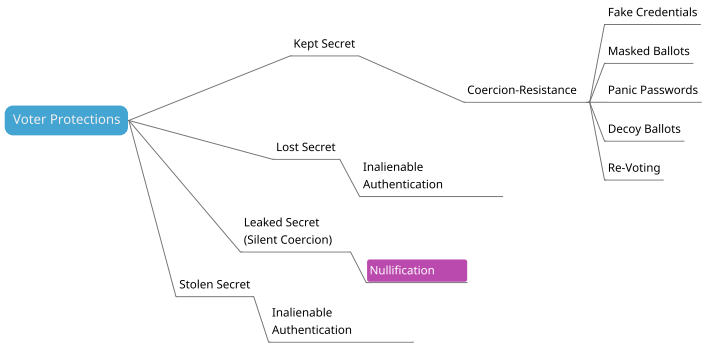

Fig. 1. Idea maze for protecting voters from coercion and credentials theft. Idea maze for protecting voters under various forms of credential loss or coercion. Nullification includes both passive approaches (*e.g.,* Caveat Coercitor) and our new active mechanism.

1.1 Ballot Privacy is not Always Enough

Definitions of ballot privacy roughly state the following: an adversary with access to the complete cryptographic transcript of the election plus control over a bounded set of other voters and election officials (and knowledge of everything the corrupted parties see), and access to Alice's receipt cannot establish how Alice voted any better than if they only had access to the final tally alone.

An implicit assumption is that Alice follows the protocol with a self-interest in preserving her privacy. What if Alice sabotages her own privacy? First, consider why. If the system design allows for receipts to leak information about how they were cast, Alice might be able to take payment for casting her vote for a specified candidate, and is incentivized to have her receipt serve as proof that she complied. She could also be threatened or coerced to vote a particular way, and an adversary savvy to the details of the voting system could demand a receipt constructed according to the coercion instruction. As common in the literature, we use the term coercion resistance to mean a system defeats both vote buying and actual coercion.

A prominent example of a remote cryptographic voting system that provides basic ballot secrecy but not coercion resistance is Helios [2,3], which originally included a "coerce me" button that would spit out a proof of how the voter voted [2] (a form of critical design to educate the voter on the difference between voter privacy and coercion-resistance). Coercion resistance is challenging in a remote setting because the adversary can observe the voter cast their ballot, or more simply, take the voter's authentication credentials and vote on the voters' behalf themselves.

1.2 Coercion Resistance Mechanisms

Table 1. Five basic mechanisms from the literature for coercion resistance. The table excludes procedural in-person fallback mechanisms (*e.g.,* Swiss postal voting) in favor of purely cryptographic protocols applicable to remote settings.

Type	Example	Non-Standard Assumptions
Fake credentials	JCJ (2005) [24]	Voter maintains at least one secret (real private key) from coercer and registers prior to coercion. Voter can simulate zero knowledge proofs to match lies to coercer. Voter knows of one honest election trustee. Under coercion, voter can always vote their intention but requires a moment alone (any time before or after coercion).
Masked ballots	WeBu09 (2009) [33]	Voter maintains at least one secret (integer offset) from coercer and registers prior to coercion. Voter can perform arithmetic in head before lying to coercer. Under coercion, voters can at least spoil their ballot.
Panic passwords	Selections (2011) [12]	Voter maintains at least one secret (real password) from coercer and registers prior to coercion. Voter can make up a fake password before lying to coercer. Under coercion, voter can always vote their intention but requires a moment alone (any time before or after coercion).
Decoy ballots	RS-Voting (2012) [7]	Not all voters have ballots and voters can opt to receive a decoy ballot that will not be counted. Voter registers prior to coercion. Under coercion, voters can use a decoy or deny having a real ballot.
Re-voting (with E2E)	VoteAgain (2020) [26]	Voter maintains an authentication mechanism that cannot be duplicated by the coercer (or eliminated). Voter must submit a corrective ballot after coercion. Assuming this, voter can always vote their intention.

Many papers have examined a relatively small set of approaches to providing coercion resistance (see Table 1), which we assume are overlayed on an E2E cryptographic voting scheme that provides ballot privacy and tally correctness.

Four of the approaches in Table 1 are based on faking information (credentials, masks, panic passwords, and decoy ballots) to give to the coercer. Decoy ballots can be distinguished by providing a game-theoretic solution (voters can request fake ballots from the EA and use these to flood the market, driving down the economic feasibility of coercion/vote selling) rather than a cryptographic one (although cryptography is used in other aspects of the system). The remaining three make similar cryptographic assumptions about voters being able to keep secrets and deceive the coercer.

The other approach, re-voting, enables votes to be updated over time, counting only the last ballot. Under coercion, a voter allows the adversary to cast their ballot for them, but then secretly changes it after. This mechanism is less rigorous than the fake ballot mechanism as the adversary can simply ensure the voter does not have the ability to change their vote after coercion (by supervising the voter until the election ends). For this reason, re-voting is often offered in a hybrid remote/in-person model. Voters vote online optimistically and then can correct coercion in-person as necessary.

Table 2. Types of key loss. Traditional coercion-resistant voting strategies consider the first row ("normal conditions"), whereas we argue all rows of key loss (see Eyal [16]) should be addressed.

Type	User has key?	Adversary has key?	Countermeasure
Normal conditions	Yes	No	Coercion resistance
Lost secret	No	No	Inalienable authentication
Leaked secret	Yes	Yes	Nullification
Stolen secret	No	Yes	Inalienable authentication

1.3 Relationship to Authentication

It is maybe not be apparent but coercion resistance is related directly to authentication. As pointed out by Hirt and Sako [21,22], a basic axiom of coercion resistance is: if the adversary has/knows/is everything that the voter has/knows/is, the voting system cannot distinguish the voter from the adversary, and therefore cannot provide a mechanism to register the true voter's vote and not the adversary's. Reconsider the coercion-resistance mechanisms above and see that they rely on a secret value or authentication mechanism that can still distinguish a voter under coercion. This is also why most coercion resistance mechanisms assume the voter can register prior to coercion, otherwise the adversary can coerce the voter to register with authentication values only the coercer knows.

For fully online voting, maintaining secrets is particularly challenging because digital forms of secrets are generally "something you have/know" rather than "something you are," and secrets can be stolen through a variety of covert means, including malware and physical access to the voter's device. Directly stealing a secret side-steps known coercion-resistant mechanisms—each assumes somewhere that the voter knows of, and possibly plays a role in, resisting coercion (such as providing a fake secret or by voting again with their real secret).

We have used the term "stolen" loosely. In fact, there are three specific failures (see Table 2) a user can experience in maintaining a secret value [16]: secrets can be *lost*, where neither the user or adversary has the credential; *leaked*, where the user and adversary both have the credential; or *stolen*, where the user loses the credential and the adversary gains the credential (so only the adversary has the credential).

For lost and stolen secrets (Rows 2 and 4), the voter loses their secret. Our countermeasure is to ensure voters cannot lose their secret through *inalienable authentication*, which is a protocol where the voter proves that they have committed their secret to their own human memory without revealing the secret.

The threat of a leaked credential (Row 3) is under-appreciated in the voting literature. It is easy to think that coercion resistance already covers this attack; however, coercion resistance addresses only *overt* attempts by a coercer to learn the key, whereas we also need to consider *covert* attacks for which voters may be unaware. Grewal *et al.* [18] term leaked credentials as "silent coercion." The reader might not consider this "coercion" as there is no pressure or threat,

however we do not attempt to adjust the terminology already accepted in the voting literature.

1.4 Silent Coercion

It is easy to think Row 3 is an unsolvable problem: if the adversary knows every secret the voter knows, the remote system cannot distinguish the voter and adversary. The truth of this unpleasant fact implies that a dimension of the problem indeed is unsolvable: the voter cannot always cast their true intent and prevent the adversary from doing the same. Nevertheless, the voter still has a course of action: the voter (or either party) can sabotage (*nullify*) the vote, preventing themselves and the adversary from casting a ballot.

Caveat Coercitor [18] is an extension to JCJ [24]. Like JCJ, it provides coercion resistance through fake credentials. However it adds two extensions: (1) any vote cast multiple times with the same credential but different candidates (in at least one occurrence) is discarded from the tally, and (2) the number of discarded ballots is made public as an indicator of the level of coercion that was present in the election. Caveat Coercitor has two main drawbacks. First is the complexity of the election trustees tabulating the tally. It inherents the quadratic (in the number of cast ballots) complexity of JCJ and the coercion evident extension adds a linear work making it cubic. As JCJ is considered too slow for practical election sizes already [13], Caveat Coercitor is mainly a theoretical advancement. The second drawback is more minor but Caveat Coercitor cannot compose easily with all the coercion resistant mechanisms in Table 1—it works with fake credentials and perhaps others but it cannot work with revoting, as voters who change their mind are excluded from the tally.

We argue there is room for more work on nullification and exploring different approaches might be worthwhile. This paper presents a different design that sidesteps these two drawbacks of Caveat Coercitor—our extension (a) is mostly linear with one quadratic component that can run concurrently [15] over the nullification period and (b) our extension can compose with revoting (along with fake credentials and panic passwords).

Our design also differs from Caveat Coercitor where nullification is an active task taken by a voter (or an enlisted helper) in responses to a ballot that does not reflect their voting intent. The passive nullification of Caveat Coercitor removes a task for the voter, likely improving usability, however our active nullification offers finer grain control: it can be triggered on an adversarial ballot even if the voter abstained from voting, and it can be not triggered in the case of a revote where the voter changes their mind. Whether more control through more decisions is a net benefit for voters will likely divide researchers, but we think it is worth laying out multiple options in the academic literature.

1.5 Other Examples of Nullification

Other instances of nullification appear in voting systems that allow voters to submit encrypted deltas to neutralize or adjust their previous votes. In the masked

voting approach of Wen and Buckland [33], each voter receives a secret mask and casts a ballot encrypted as the difference between their vote and this mask. A coerced voter can later submit a masked ballot that offsets their original vote, effectively nullifying it in the additive tally. A similar pattern occurs in the update scheme of Kulyk, Teague, and Volkamer [25], where a voter's row on the *bulletin board (BB)* is multiplicatively combined from multiple encrypted values. To cancel their vote, a voter can submit the multiplicative inverse of their earlier choice, neutralizing its effect on the final tally. These schemes provide important protections against overt coercion or vote selling, where the voter is aware of and responding to coercion. However, they do not address silent coercion scenarios, where an adversary obtains the voter's credentials and casts a ballot without the voter's knowledge. In other words, while these are examples of nullification, they are not designed to prevent or recover from undetected adversarial voting, as in Caveat Coercitor [18] and our work.

As an additional remark, the cryptographic structure of our nullification protocol (see Sect. 3.1) parallels the proof techniques used by Kulyk, Teague, and Volkamer [25]. In both cases, the core idea is to construct a zero-knowledge proof that supports a disjunctive claim—either the value was submitted by an eligible voter who knows the signing key or the value has no effect (*e.g.,* add a 0 or multiply by 1).

1.6 Our Contributions

We provide a protocol for *nullification* that can work in concert to resist silent coercion where voter keys or secrets are leaked to the adversary.

The mechanism is an overlay or add-on for coercion resistant voting system and does not compete with any of the mechanisms in Table 1. In fact, even within the table, the basic mechanisms are complimentary to some extent (*e.g.,* an E2E voting system might use panic passwords and revoting and decoy ballots). The basic idea of our paper is that if a voter is not coerced, they vote as normal; if they are coerced but their key is not leaked, they use a mechanism from Table 1; if their key is leaked, they use nullification which is better than doing nothing (see Fig. 1).

With this said, we want to be clear that our nullification protocol cannot be simply glued onto Helios or JCJ or Selections. Our protocol expects a certain interface to work, and some additional changes to the voting system can greatly enhance its usability and safety. For this reason, we do sketch a basic voting system (VoteXX) to demonstrate how a voting system can compose with nullification.

2 Preliminaries

We adopt a number of common cryptographic primitives from the voting literature. All operations are performed in the same DDH-hard elliptic curve group. Digital signatures are performed with the Schnorr signature scheme. Encryption

is performed with exponential Elgamal [14], which can be augmented with *distributed key generation (DKG)* and threshold decryption (for m out of n key holders [31]). As standard in the voting literature, we assume the EA has a threshold public key and do not collude beyond the threshold. We use standard $\Sigma-$Protocols to prove knowledge of discrete logarithms (Schnorr [32]), knowledge of representations (Okamoto [29]), and knowledge of Diffie-Hellman tuples (Chaum-Pedersen [11]), which also corresponds to ElGamal re-randomizations and decryptions. We also use techniques to allow the trustees to compute jointly, verifiably (*i.e.,* produce $\Sigma-$Protocol proofs), and privately on ElGamal ciphertexts the following: (i) a random shuffle of ciphertexts [28], and (ii) the evaluation of an *binary gate* operations based on their logic lookup table (mix and match [23]).

3 New Nullification Protocol

We explain our nullification protocol by stating its front-end requirements and introducing the semi-trusted "hedgehog" agents.

3.1 Front-End Requirements

Our nullification protocol is an overlay, add-on, or "back-end" for the "front-end" of a E2E coercion resistant voting schemes. As a backend, nullification adds protection for Row 3 in Table 2 to systems that already protect against Row 1. However none of these front-ends will "just work" with nullification, they need to be modified to compose correctly. For space, we cannot fully detail how to add nullification to every scheme however we will sketch how to do it with JCJ [24] and Selections [12], as well as showing a new front-end that adds an additional feature (called hedgehogs). JCJ is based on fake credentials, and Selections allows re-voting in addition to panic passwords, so Selections as augmented below actually provides three lines of defence against coercion (panic passwords, re-voting, and nullification).

The details of E2E voting systems vary widely but we assume a basic architecture. Registration phase: voters register for the election which may or may not post registration data to a BB—an append-only, non-equivocating broadcast channel. Voting phase: voters cast ballots which do not reveal how they voted and are posted on the BB. Tallying Phase: the votes on the BB are used to produce a final tally. Finally we introduce a Nullification Phase: voters (or their agents) have a window of time to inspect the cast ballots and potentially nullify adversarial ballots submitted using their key.

In JCJ, a ballot from the voter comprises their vote, the asserted credential (secret key) of the voter (we place this in the exponent to make it easier to compare side-by-side with Selections), and some zero knowledge proofs of correct formation that we omit: $\left\langle [\![\mathsf{vote}]\!], [\![g^{\hat{\mathsf{sk}}}]\!] \right\rangle$. In Selection, a ballot consists of $\left\langle [\![\mathsf{vote}]\!], g^{\hat{\mathsf{sk}}}, [\![\mathsf{pk}]\!] \right\rangle$ where $\hat{\mathsf{sk}}$ is the asserted password of the voter, and $[\![\mathsf{pk}]\!]$ is the

entry from the voting roster (created at registration time). If $g^{\hat{\mathsf{sk}}} = \mathsf{pk}$, the vote is tallied, otherwise it is discarded (later after anonymization).

We now consider the front-end requirements to work with our nullification back-end:

1. The voter's eligibility to vote is determined by knowing a secret key,
2. The voter can see how they voted from the BB, and
3. The voter can add a secret "flag" to their own cast ballot(s) that can be used as a filter after the ballots are anonymized.

The first requirement is already true of must coercion resistance schemes, including those based on fake credentials [24], panic passwords [12], and re-voting [26]. The second requirement sounds like a strange requirement because, until now, the literature assumes voters are posting their own ballots or in the case of coercion, the voter at least knows coercion is happening. But in our threat model of leaked keys (Row 3 of Table 2), an adversary might vote with a voter's key and the voter is not unaware and, in fact, has no way to tell from the BB because the ballot components are all encrypted under the EA's public key. This is also true of hybrid systems (Internet voting with in-person fallback) where a voter might be able to distinguish their actions from the adversary by attending in-person with government photo identification (or similar).

The fix for JCJ and Selections is to require a voter to encrypt every ballot twice: once under the EA's key and once under the voter's key. The voter must then add a proof (using a $\Sigma-\mathsf{Protocol}$) that: (i) the two encryptions are of the same value (standard plaintext equality) and (ii) the public key used in the second encryption matches the asserted voter credential in the ballot (possible using Mauer's abstraction [27]). In JCJ, voters looking for other ballots cast with their keys will use trial decryption on every ballot to see if any belong to them. In Selections, the ballot includes the password (real or panic) in a deterministic commitment (rather than a randomized encryption) so it is simpler: the voter checks if a commitment to their real password ever shows up in a submitted ballot they did not actually submit. If they find one, they can decrypt the ballot and take an action: leave it if it is for their candidate, re-vote if there is time left, and nullify as a last resort if voting has already closed.

The third requirement is that a flag is added by the voter. The voter will construct a "selector" vector of encrypted values, with one value for every ballot submitted during the voting phase. For example, if there are 5 ballots and the voter wants to flag the 3rd ballot, the vector could be: $\langle [\![0]\!], [\![0]\!], [\![1]\!], [\![0]\!], [\![0]\!] \rangle$. In this example, the unflagged value is 0 and the flagged value is 1. If a voter flags a ballot with ($[\![1]\!]$), it must know the private key used in the construction of the ballot; otherwise, any voter could nullify any vote. However, when the voter submits a false flag ($[\![0]\!]$), it does not need to know the associated key. To enforce these constraints, the voter 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 sk corresponding to the pk used in a component of the ballot)].

In Case 1, the voter 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 $y = g^{\hat{sk}}$, 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 voter 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 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 $\Sigma-\mathsf{Protocol}$ due to Schnorr: $\langle a_C, e_C, z_C \rangle$. To summarize, the hedgehog proves: $\Pi := [\text{A } \texttt{OR } (\text{B } \texttt{AND } \text{C})]$ for each flag.

Further, the resulting proof can be made non-interactive (typically in the random oracle model with the Fiat-Shamir heuristic [17], in its strong form [4], but other heuristics exist [20]). 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 $\langle a_C, e_B, z_C \rangle$.

The final consideration is where in the tallying process are the flags used? This will be specific to the voting system. For JCJ and Selections, a suitable design is to add flags to ballots before tallying. As nullification vectors arrive, the EA can flatten them into a single vector that is the logical AND of all vectors, using a binary lookup and the Mix and Match protocol [23]. Probably the simplest way to nullify a ballot that has been flagged is to just replace the key used to cast the ballot with a "junk" key value such as 0, again using a lookup table (outputs are always rerandomized).

$$[\![g^{\hat{sk}}]\!] \leftarrow \begin{array}{c|c} \textbf{Flag} & \textbf{Out} \\ \hline [\![0]\!] & [\![g^{\hat{sk}}]\!] \\ [\![1]\!] & [\![0]\!] \end{array} \tag{1}$$

However a special purpose trick can be used as a shortcut when the discrete logarithm is hard and the encryption is additively homomorphic (both are true in JCJ and Selections): a beacon selects a random exponent rand after voting closes, nullification uses $[\![0]\!]$ as a non-flag and $[\![\mathsf{rand}]\!]$ as the flag, and then simply: $[\![g^{\hat{sk}}]\!] \leftarrow [\![\mathsf{flag}]\!] \cdot [\![g^{\hat{sk}}]\!] = [\![\mathsf{flag} + g^{\hat{sk}}]\!]$. When the flag is $[\![0]\!]$, $[\![g^{\hat{sk}}]\!]$ is unmodified and when it is $[\![\mathsf{rand}]\!]$, $[\![g^{\hat{sk}}]\!]$ is "ruined" by casting it to an arbitrary integer (that is unpredictable at the time $[\![g^{\hat{sk}}]\!]$ is submitted to the BB) and the secret key can no longer be extracted (due to the discrete logarithm problem). Since the same ballot can be nullified an arbitrary number of times, we assume some system-level restrictions limit an adversary that nullifies so much, the value returns to its original $[\![g^{\hat{sk}}]\!]$ (an *integer overflow* attack). These methods effectively turn any real credentials into fake credentials (or real passwords into panic passwords). Since the voting system will already eliminate (in a privacy-preserving fashion) fake credentials, the tally can be computed as normal after this pre-processing step.

The general approach (using lookups) is expensive: each nullification request requires work linear in the number of ballots, making it quadratic overall. However all the lookup tables can be pre-computed concurrently with the nullification phase of the election, and the flattening (logical AND) can be done immediately by the EA once it receives a new nullification request. Assuming the EA can keep up nullification requests, only an additional linear amount of work is needed to be done at tallying time. The special purpose trick is fast for the EA: it requires only multiplications and no exponentiations.

Additional Remarks. Nullifications are not individually revealed on the BB, but the difference between the provisional and final tallies provides coercion evidence analogous to that in Caveat Coercitor. A coercer might attempt to verify whether a coerced vote was nullified by casting a vote using the voter's credential and later checking whether a nullification occurred. Our protocol, however, does not reveal nullification events on a per-ballot or per-credential basis. Ballot nullifications are included only in the transition from the provisional to the final tally, and the nullification request itself is either hidden (via cryptographic flagging) or aggregated. In particular, a coercer who holds the voter's credential cannot determine whether a nullification occurred.

The JCJ definition of coercion resistance includes the ability to avoid forced abstention attacks [24]. In the passive nullification of Caveat Coercitor [18] and the active nullification of our work, a voter can cryptographically prove that the ballot associated with a specific credential was nullified. To evade forced abstention, nullification needs to be composed with a coercion resistance mechanism (see Fig. 1). For example, with fake credentials, the voter can prove credentials are nullified but cannot prove the credential is real. A promising direction for future work is to develop deniable nullification mechanisms where no individual voter can prove that they nullified a ballot.

3.2 Hedgehogs

We will present one final idea: can we allow voters to enlist helpers to help them with nullification? This is useful for voters who are not savvy about coercion, which likely overlap with voters likely to leak their keys unintentionally. The idea is to let a voter prearrange with a helper: "I want to vote for Alice and this is the key I will use; if you see a vote for Bob on the BB with my key, please nullify it."

This works if the helper is *fully trusted*. Precisely we are making two trust assumptions: (i) The helper will nullify if there is a vote for Alice, and (ii) the helper will not nullify if there is a vote for Alice (which it has the power to do). The first assumption can be addressed by enlisting many helpers, hoping that at least one will act. The second assumption is difficult to fix. However we will now propose a new front-end voting system that does not require the second trust assumption. In other words, helpers are *semi-trusted*: they are trusted to

Table 3. The threat model for different actors in an election with a blue and red candidate. The voter prefers the blue candidate and the adversary the red candidate. The symmetry between the adversary with the leaked credentials and the voter shows why the voter cannot register their true intent reliably. The asymmetry between the voter and the hedgehog is the contribution of the VoteXX front-end.

Action	Blue Voter	Red Adversary (no credentials)	Red Adversary (with both credentials)	Blue Hedgehog (blue credential only)
View bulletin board	Yes	Yes	Yes	Yes
Cast a blue vote	Yes	No	Yes, but would not	Yes
Cast a red vote	Yes, but would not	No	Yes	No
Nullify voter's blue vote	Yes	No	Yes	No
Nullify adversary's red vote	Yes	No	Yes	Yes

Voting.
Each voter completes voting online. We assume voters have already registered two keys or passphrases: one used to vote YES and one for NO. 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: $\sigma_{\mathsf{yes}} \leftarrow \mathsf{Sig}(\mathsf{nonce})$. 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 an anonymous channel to the BB. The EA marks it as invalid if it is an exact duplicate or if the proofs are invalid.

Protocol 1: Voting Protocol.

actually act when they are supposed to, but they cannot act against the voter's wishes. We call semi-trusted helpers *hedgehogs*.

Hedgehogs are introduced not because they possess capabilities that voters lack in principle, but because they are more reliable agents for acting on those capabilities in practice. While it is true that a technically aware voter could inspect the BB and identify unauthorized ballots submitted using their credential, most voters are unlikely to do so—especially if they are unaware that their key has been compromised. Hedgehogs serve as dedicated monitors who scan the BB for suspicious ballots and can proactively initiate nullification before the voter ever notices. Hedgehogs might be benign election watchdogs or partisan parties who are selected by voters who share the same political allegiance. For

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

1. The EA takes the list of $\langle [\![\mathsf{pk}]\!], [\![\sigma]\!] \rangle$, verifiably shuffles them, 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.

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 $\langle \mathsf{pk}_{\mathsf{yes}}, \mathsf{pk}_{\mathsf{no}} \rangle$. 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 $\mathsf{yesVotes}$). 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. $\mathsf{yesVotes}$ 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 $\mathsf{yesVotes}$. 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 $\mathsf{yesVotes}$ and computes its logical AND under encryption using the mix and match secure function evaluation (SFE) protocol [23]. It repeats this process for the remaining $\mathsf{pk}_{\mathsf{no}}$ keys.
2. The EA takes the list of encrypted OR-ed flags, sums them under encryption, and verifiably 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 $\mathsf{noVotes}$.

Protocol 2: Tallying Protocol (including nullification).

example, each political party or candidate could offer a hedgehog to their voters (Table 3).

Our approach is to redesign a front-end voting system (we call VoteXX) that uses multiple keys to identify voters, instead of just one (as in JCJ and Selections), where the keys are tied to specific candidates in the election. We will use the example of two keys: $\mathsf{pk}_{\mathsf{yes}}$ to vote yes and $\mathsf{pk}_{\mathsf{no}}$ to vote no. The front-end is based on signature-based mix network voting, *cf.* [6,19,30]. The nullification period runs after a provisional tallying phase. 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. Thus voters who want to vote yes will provide the hedgehog with only $\mathsf{sk}_{\mathsf{yes}}$ and will not provide $\mathsf{sk}_{\mathsf{no}}$.

VoteXX with nullification is given in Protocols 1 and 2. VoteXX is not necessarily better or worst than augmenting JCJ or Selections with nullification, it just shows the flexibility of nullification and illustrates different trade-offs that are possible:

- **Hedgehogs:** In VoteXX, helpers are only semi-trusted while in JCJ/Selections they are fully trusted.
- **Normal coercion-resistance:** In VoteXX, there is no support for normal coercion resistance (when the adversary does not know the key), while JCJ offers fake credentials and Selections offers both panic passwords and re-voting.
- **Multiple candidates:** In VoteXX, each additional candidate in the election requires an additional inalienable passphrase to be memorized by the voter. In JCJ/Selections, the requirements on the voter does not change with the number of candidates.
- **Provisional Tally:** In VoteXX, a provisional tally is published and then adjusted based on nullification, which reveals early information about the tally. In JCJ/Selections, nullification is done prior to tabulating the tally.
- **Efficiency:** In VoteXX and Selections, tallying is linear without nullification while JCJ is quadratic without nullification.

4 Variants and Extensions

Vote Flipping. Our design supports a variety of options for implementing the semantics of nullification, including what we call "cancel" or "flip." We recommend cancel, which is the default and is what has been described thus far. 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. 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 $x+y \bmod 2$, 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 2: `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). We view flip not as re-voting, but as "randomizing" the vote, which is a form of nullification.

As we point out below, under stronger assumptions, there are some use cases in which flip can be used to re-vote. Also, nullification can be used as an overlay in conjunction with re-voting strategies. 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. The spouse can later flip their vote to Bob without the coercer knowing.

Coercion Evidence. Our system also supports a form of coercion evidence, though in a different style from Caveat Coercitor [18]. Rather than embedding an explicit coercion-evidence test in the tallying process, we allow coercion evidence to be derived by computing the difference between the provisional tally (before nullification) and the final tally (after nullifications are processed). This difference quantifies the effect of adversarial interference, credential leakage, or voter-initiated ballot cancellations, and can be used by observers to assess the integrity of the election outcome.

Inalienable Authentication. A related but orthogonal direction to nullification is inalienable authentication (recall Fig. 1): ensuring that only the voter—rather than their device—possesses the factor needed to cast a valid vote. The idea traces back to Achenbach et al. [1] and is further discussed in ecash 2.0 [10], where a credential is bound not to a device but to a secret memorized by the user. This offers a path to prevent adversaries from voting even when they control the voter's device or possess all stored credentials. However, while prior work assumes such a mechanism exists, no concrete implementation has been adopted in remote voting systems. We view this direction as promising but currently underdeveloped, particularly in terms of usability and resistance to coercion.

There remain substantial challenges in designing usable inalienable authentication. A voter must prove knowledge of a secret without revealing it, and without offloading trust to a potentially compromised device. This raises questions about entropy, memorability, leakage, and fallback. As a conceptual example, a voter could demonstrate knowledge of a passphrase by computing a blinded linear function over a challenge (e.g., $c \cdot x + r$), using a physical aid such as a printed wheel. Though primitive, this illustrates that inalienable authentication has not been fully explored as a human-verifiable building block for remote voting. We leave the development of practical, zero-knowledge, and simulatable versions of this primitive to future work.

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 [34].
Malware Protection. 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.

5 Conclusion

Nullification offers a response to credential compromise in voting systems that are otherwise coercion resistant. While our protocol puts forward a concrete proposal, the broader design space remains far from settled—especially around the practicalities of hedgehogs; the composability of nullification with existing mechanisms like panic passwords, re-voting, and fake credentials; and designs enable a response to silent coercion without imposing unrealistic memory, attention, or procedural demands.

References

1. Achenbach, D., Kempka, C., Löwe, B., Müller-Quade, J.: Improved coercion-resistant electronic elections through deniable re-voting. USENIX JETS (2015)
2. Adida, B.: Helios: web-based open-audit voting. In: USENIX Security Symposium, pp. 335–348 (2008)
3. Adida, B., Marneffe, O.d., Pereira, O., Quisquater, J.J.: Electing a university president using open-audit voting: analysis of real-world use of Helios. In: EVT/WOTE (2009)
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. Carback, R.T., et al.: Scantegrity II municipal election at Takoma Park: the first E2E binding governmental election with ballot privacy. In: USENIX Security Symposium (2010)
6. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–90 (1981)
7. Chaum, D.: Random-Sample Voting (2012). online
8. Chaum, D., et al.: Votexx: a solution to improper influence in voter-verifiable elections. In: E-Vote-ID (2022)
9. Chaum, D., et al.: Votexx: extreme coercion resistance. IARC ePrint 2024/1354 (2024)
10. Chaum, D., Moser, T.: ecash 2.0 inalienably private and quantum-resistant to counterfeiting. Tech. rep. (2022)
11. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO (1992)
12. Clark, J., Hengartner, U.: Selections: internet voting with over-the-shoulder coercion-resistance. In: Financial Cryptography (2011)

13. 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)
14. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: EUROCRYPT (1997)
15. Essex, A., Clark, J., Hengartner, U.: Cobra: Toward concurrent ballot authorization for Internet voting. In: EVT/WOTE (2012)
16. Eyal, I.: On cryptocurrency wallet design. In: Tokenomics (2021)
17. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: CRYPTO, pp. 186–194 (1986)
18. Grewal, G.S., Ryan, M.D., Bursuc, S., Ryan, P.Y.A.: Caveat coercitor: coercion-evidence in electronic voting. In: IEEE Symposium on Security and Privacy (2013)
19. Haenni, R., Spycher, O.: Secure Internet voting on limited devices with anonymized DSA public keys. In: EVT/WOTE (2011)
20. Hazay, C., Lindell, Y.: Search problems. In: Efficient Secure Two-Party Protocols. ISC, pp. 227–254. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14303-8_9
21. Hirt, M.: Multi-Party Computation: Efficient Protocols, General Adversaries and Voting. Ph.D. thesis, ETH Zurich (2001)
22. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: EUROCRYPT (2000)
23. Jakobsson, M., Juels, A.: Mix and match: Secure function evaluation via ciphertexts. In: ASIACRYPT (2000)
24. Juels, A., Catalano, D., Jacobsson, M.: Coercion-Resistant electronic elections. In: ACM WPES (2005)
25. Kulyk, O., Teague, V., Volkamer, M.: Extending helios towards private eligibility verifiability. In: E-Voting and Identity (2015)
26. Lueks, W., Querejeta-Azurmendi, I., Troncoso, C.: Voteagain: A scalable coercion-resistant voting system. In: USENIX Security (2020)
27. Maurer, U.: Unifying zero-knowledge proofs of knowledge. In: AFRICACRYPT (2009)
28. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: ACM CCS (2001)
29. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: CRYPTO (1992)
30. Park, C., Itoh, K., Kurosawa, K.: Efficient anonymous channel and all/nothing election scheme. In: EUROCRYPT (1993)
31. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: EUROCRYPT (1991)
32. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991). https://doi.org/10.1007/BF00196725
33. Wen, R., Buckland, R.: Masked ballot voting for receipt-free online elections. In: VOTE-ID (2009)
34. Zagórski, F., Carback III, R.T., 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)