Cobra: Toward Concurrent Ballot Authorization for Internet Voting

Aleksander Essex
Children's Hospital of Eastern Ontario Research Institute

Jeremy Clark
Carleton University

Urs Hengartner
University of Waterloo

Abstract

We propose and study the notion of concurrent ballot authorization for coercion-resistant, end-to-end verifiable (E2E) internet voting. A central part of providing coercion resistance is the ability for an election authority to filter out fake ballots from legitimate ones in a way that is both private and universally verifiable. This ballot authorization process, however, can potentially come at a heavy computational cost. In previous proposals, the bulk of this computation cannot be performed until the last ballot has been cast. By contrast, concurrent ballot authorization allows ballots to be authorized as they are submitted, allowing the tally to be declared immediately after polls close. An efficient tally is especially important in the coercion-resistant internet voting setting, as it is particularly vulnerable to denial of service attacks caused by floods of fake ballots.

We present a proof-of-concept voting system, Cobra, the first coercion-resistant system to offer concurrent ballot authorization. Although Cobra offers the fastest tallying relative to the related work, it has a registration process that we consider to be too slow to be viable; one that is quadratic in the number of eligible voters. We present Cobra as a first-step toward what we hope will become a standard feature of coercion-resistant internet voting schemes: concurrent ballot authorization.

1 Introductory Remarks

Internet voting is a hard problem. Out of any way to cast a ballot, it arguably demands the strongest adversarial model. Casting a ballot online subsumes all the problems of casting a ballot in-person (integrity and ballot secrecy) and by mail (in-person coercion, vote buying and selling, and secure transport), plus it requires voters to submit their secret ballots from potentially infected personal computers over a hostile network for storage on an internet-facing server.

In the face of these overwhelming odds, researchers have done well at addressing these problems individually. In this paper, we focus on the problems of coercion and vote selling. The lack of a private booth in internet voting means anyone can potentially observe you as you vote or cast a ballot on your behalf. This is addressed in the literature, starting with Juels *et al.* [32], by providing each voter with an authentication credential and the ability to generate fake credentials that are indistinguishable from real ones. If coerced, the voter provides a fake credential and later covertly votes with their real credential. If a voter wants to sell their vote, they have no way to prove that the credential they are providing is indeed their real one, even if they want to. These properties hold even if the voter is corrupted before obtaining their credential (during a phase called *registration*).

Using cryptographic techniques, fake ballots can be verifiably separated from legitimate ballots without ever revealing whether a particular submission was real or fake. We call this process *ballot authorization*. Ballot authorization, however, typically is a computationally intensive process, and must be performed before the ballots can be tallied. To make matters worse, it has been a process that, for the most part, must occur after all of the ballots in an election have been submitted.

Ballot Authorization Function. The intuition behind concurrent ballot authorization is straightforward. The trustees of an election authority engage in a secure, universally verifiable protocol implementing a ballot authorization function. This function is applied to each ballot individually, resulting in one of two (indistinguishable) outcomes: If the ballot credential is valid, the function preserves the encrypted vote. Conversely, if the credential is invalid, the function replaces the encrypted vote with an encrypted non-vote (e.g., 0 in the case of a homomorphically tallied election). Realizing this ballot authorization functionality is the focus of this paper.

Contributions. Our primary contributions are as follows:

1. We introduce the notion of concurrent ballot authorization for coercion-resistant internet voting, allowing ballots to be authorized as they are being submitted.
2. Through concurrent ballot authorization, we offer a novel technique to mitigate the impact of board flooding [33]—a type of denial-of-service attack to which coercion resistant elections are fundamentally vulnerable.
3. We present a proof of concept system Cobra[1] offering the fastest time-to-tally relative to previous work.

2 Preliminaries

Coercion-Resistance. To define coercion-resistance, consider the following game [36]. A voter is selected by the adversary to be coerced (prior to registration). Let the voter's true voting intent be γ_v, *e.g.,* to vote for Alice.[2]

Within the game, a coin is flipped. If the coin flip is heads, the voter will comply with the coercer. This means the voter truthfully tells the coercer anything he asks and follows his instructions. If the coin flip is tails, the voter will try to deceive the coercer. She will give him fake information and when she is instructed to do something, she will act as if she did so, although she may not necessarily have complied.

Informally, we say a voting system is coercion-resistant with respect to γ_v if two properties hold: (1) if the coin flip is tails, the voter can always accomplish γ_v[3] and (2) the coercer cannot guess whether the voter is complying or deceiving him with more (or less) success than he could if he played the same game with an ideal system where voters give their votes to a trusted party and the trusted party produces the correct tally.

Registration in Coercion-Resistant Schemes. Most coercion-resistant internet election systems inherit the architecture of the first such system, due to Juels *et al.* [32] (JCJ). Well prior to the election, each voter must register with a set of Registrars. To be coercion-resistant, at least one Registrar must not collude with the adversary, the voter must know which one is trusted [32, 16], and the adversary cannot even passively eavesdrop on the communication with this Registrar [35] (*e.g.,* communication is over an untappable channel). The elimination of all untappable channels appears impossible for coercion-resistant voting [26]. To simplify things, we assume there is a single Registrar and registration is done privately in-person.[4]

The registration process outputs: a credential (*e.g.,* a cryptographic key) to each voter, an encryption of each credential posted on a public Roster, and a designated verifier proof [30] given to each voter that their posted encryption is correct. If coerced into providing their credential, the voter can give the adversary a fake cryptographic key and a simulated proof that this fake key is what is on the Roster.

We follow a few modifications of this model as made by Clark and Hengartner with Selections [15]: first, a credential is a voter-selected password from a panic password scheme [14]. Second, registration does not rely on designated verifier proofs. Together, this absolves the voter from having to prepare for the possibility of coercion by computing fake values and transcripts. Instead, to evade coercion, a voter simply has to create a spurious password, which can be done mentally on-the-fly.

Ballot Authorization & Tallying. Ballot submission in the JCJ architecture is anonymous and open to anyone. Were it not, a coercer could tell if a voter he coerced submitted a second ballot with a different (likely correct) credential. Each voter submits a well-formed ballot and an obfuscation of an credential (either real or fake). The election authority must be able to systematically filter out (1) ballots that are not well-formed, (2) all-but-one ballot per credential (*e.g.,* the most recent), and (3) all ballots with a credential that is not on the Roster. We call this filtering process *ballot-roster authorization* or *ballot authorization* for short (others have called it *vote authorization* [44]).

Of these, ballot authorization is the most difficult. As part of coercion-resistance, authorization is done cryptographically such that no one learns which submitted ballots have fake credentials, while at the same assuring everyone the authorization was performed correctly. The output of ballot authorization is the subset of submitted ballots that are well-formed, each originating from a unique, registered, voter. The authority can then tally the ballots by mixing them and decrypting (if they are not already mixed by the ballot authorization process), or by

[1] Cobra: Concurrent Ballot–Roster Authorization.

[2] γ_v is expressible in general terms, such as a vote for anyone but Alice, a vote for a random candidate, or an abstention from voting.

[3] We consider systems that are coercion-resistant with respect to casting a vote according to any probability distribution across the possible candidates (including no candidate). If we let γ_v be *anything*, it could be, *e.g.,* to "vote the same way as the voter before me," which is in obvious violation of other security properties a system should provide.

[4] This may raise the question of why, if voters must register in-person, we do not simply require them to vote in-person. There are two reasons: (1) registration can be conducted over a longer period of time and (2) many systems can bootstrap a single registration into voting securely in an arbitrary number of elections (for Cobra, this is future work).

homomorphically adding them and decrypting the sum.

Compromises for Linear Tallying. The computational complexity (in terms of modular exponentiations) of the ballot authorization function in JCJ is quadratic in the number of submitted ballots. Much interest has been shown for making it linear (see Related Work in Section 5). Of the successful attempts, each compromises the original set of properties of JCJ/Civitas in some way. The Araujo *et al.* systems [5, 6] lack the ability to efficiently remove voters from a Roster or move voters to a different Roster. Selections [15] and SHKS [44] leak a small amount of information about how many times a voter has voted, which requires additional assumptions on adversarial uncertainty to provide coercion-resistance. Spycher *et al.* [49] add an additional trust assumption on the trustees in terms of adding dummy ballots.

Board Flooding Attacks. Even if ballot authorization can be made linear in the number of submitted ballots, it is still computationally intensive. Ballot submission must be anonymous (and thus open to anyone) as part of the coercion-resistance mechanism. An adversary could flood the election with fake but well-formed ballot submissions that would need to be processed [33] (*cf.* [19]). In the related work, the bulk of this processing cannot be started until after the last vote comes in because ballot authorization requires all the ballots to be mixed. Such an attack could significantly delay the announcement of the election results.

Cryptographic Toolkit. We work in the standard setting of a prime-order subgroup $\mathbb{G}_q$ of $\mathbb{Z}_p^*$ where DDH is hard (until Section 4.3). We denote a CPA-secure encryption of message m as $[\![m]\!]$. All encryption is randomized (unless otherwise stated) and performed under a single public key. We assume a distributed key generation protocol DKG exists for generating the private decryption key in a shared and threshold recoverable manner [40, 23]. We further assume the encryption scheme is additively homomorphic, rerandomizable, and the plaintext space is small. Without loss of generality, the scheme could be exponential Elgamal [18].

We use several standard primitives that are common in cryptographic voting. We use Σ-protocols for proving knowledge of a discrete log, correct ciphertext rerandomization, and correct decryption [45, 13]; techniques for conjunction and disjunction of proofs [17]; techniques for diverting proofs [39, 27]; and the Fiat-Shamir heuristic to convert Σ-protocols into NIZKPs [20]. We also use plaintext equality tests [29].

Finally, we rely on secure function evaluation (SFE) in the following setting: the function will be publicly defined, the inputs to the function will be encrypted under a public key with threshold decryption, the keyshare holders will provide a *universally verifiable* proof that the function was evaluated correctly, and no individual keyshare holder can learn the output of the function. We use the Mix & Match protocol [29]. In Mix & Match, functions that can be represented in a small truth table can be efficiently evaluated. Each element of the truth table is published in plaintext and encrypted (initially with known randomness). The table is then shuffled by each keyshare holder row-wise (preserving input/output pairs) with a universally verifiable reencryption mix network. To evaluate the function on an encrypted value, the keyshare holders perform a plaintext equality test between the encrypted input to the function and each encrypted input in the shuffled truth table until a match is found; the corresponding encrypted output is returned.

Bloom Filter. A data structure due to Bloom [8] allows for highly efficient membership testing at the cost of potential false positives. Originally conceived as a "filter" for querying high latency storage devices, a Bloom filter can store n elements in $O(n)$ space, and test for set membership in $O(1)$ operations. An important characteristic of Bloom filters is their probabilistic nature. Given a Bloom filter B and query q, a set membership test $\mathsf{Query}(q, B)$ yields the following result: $\mathbf{Pr}[\mathsf{Query}(q, B) = \textsc{True} \mid q \in B] = 1$ whereas $\mathbf{Pr}[\mathsf{Query}(q, B) = \textsc{False} \mid q \notin B] < 1$. The probability of a set membership test giving a false positive is dependent on a the bit length of the filter, the number of elements already contained in it, and the number of bits (i.e., hash functions) used to map a query to a key.[5]

3 A Template for Concurrent Ballot Authorization

In this section we discuss the high-level notion of concurrent ballot authorization and its importance. There is a natural arc to the timeline of an election: while matters typically progress slowly in the beginning, by the end there is significant interest in declaring the winner. In the timeline of an internet election, voter registration might occur on the time span of months or years. The election, *i.e.,* the ballot submission period, might occur on the time span of days or weeks. And given the electronic nature of the tally, it is a reasonable requirement that the results should be computable on a smaller timescale.

Aside from simply being a matter of convenience, having an efficient tally is especially important for coercion-

[5]In this context we mean 'key' as in *database key*, not cryptographic key.

resistant internet voting when considering the very mechanism that prevents coercion—the ability to submit arbitrarily many fake ballots. As we have already witnessed in standard internet elections, network-level denial of service attacks are a very real threat.[6] Attacks based on flooding the election authority with fake ballots [33], which are indistinguishable from legitimate ones by design, has the potential to significantly delay results-reporting if authorization cannot commence until after the election.

3.1 The Ballot Authorization Function

Ballot authorization, in essence, is a set membership test. During registration, voters add a credential authorizing them to vote, x, to a set of encrypted credentials, Roster, maintained by the election authority.

Consider a voting system where the voter submits $\langle [\![x]\!], [\![v]\!] \rangle$ in which x is the credential authorizing them to vote if $x \in \mathsf{Roster}$ and v is the vote encoded in the format required by the tallying function. Voters must also submit proof that v is well-formed, and they have knowledge of x. To realize concurrent ballot authorization, we require a function f defined as follows:

$$f([\![x]\!]) \quad = \quad \begin{cases} [\![1]\!] & x \in \mathsf{Roster} \\ [\![0]\!] & \text{otherwise.} \end{cases} \tag{1}$$

If such an f existed, we could simply take an encrypted vote $[\![v]\!]$ and credential $[\![x]\!]$, and compute under encryption $[\![v']\!] = [\![v \cdot f(x)]\!]$. This would effectively nullify ("zero-out") the vote if the credential was invalid, while preserving the vote otherwise. We call the result, $[\![v']\!]$, an authorized ballot.

The homomorphic multiplication can be accomplished directly in an encryption scheme like the one due to Boneh *et al.* [9] (BGN), which then allows the set of all authorized ballots $[\![v']\!]$ to be summed homomorphically. Alternatively, in an additively homomorphic encryption scheme like exponential Elgamal or Paillier, the multiplication can be accomplished indirectly with Mix & Match, evaluating the following truth table on $[\![x]\!]$:

In	Out
$[\![0]\!]$	$[\![0]\!]$
$[\![1]\!]$	$[\![v]\!]$

Recall that as part of the Mix & Match protocol, the outputs are shuffled and rerandomized by each trustee. Thus if $[\![v]\!]$ is the output, it will be indistinguishable from the value $[\![v]\!]$ used to construct the truth table.

[6]"Online voting company blames delays on orchestrated attempt to thwart democracy," *Toronto Star*, March 27, 2012.

All the coercion-resistant internet voting schemes we compare to (and related work on private set intersection) achieve authorization through a related function:

$$g([\![x]\!]) \quad = \quad \begin{cases} [\![0]\!] & x \in \mathsf{Roster} \\ [\![r]\!] & \text{otherwise,} \end{cases} \tag{2}$$

where r is a random value.

For example, if the Roster were an array of encrypted credentials, one could take $[\![x]\!]$ and perform a plaintext equality test between $[\![x]\!]$ and each ciphertext in the Roster [32]. For additively homomorphic encryption, a single test is equivalent to applying g. Similarly, if the Roster were an encrypted polynomial p with valid credentials as its roots [21], evaluating the polynomial at $[\![x]\!]$ is functionally equivalent to g.

At first glance, it may seem like g provides a basis for implementing f. This would require g to be modified so as to map all possible values of r to a single value. If the result of $g([\![x]\!])$ could be decrypted, then this mapping could be preformed in the clear. As part of coercion-resistance, however, the ballots must be anonymized *before* their validity is revealed, yet anonymization in this context (*e.g.,* via mixing) can only occur *after* the complete set of ballots are submitted. Conversely, implementing this mapping under encryption in an efficient and universally verifiable way for a random exponent $r \in \mathbb{Z}_q$ (or a group element) is not forthcoming from techniques in the literature. Thus for implementing f, we must consider constructions other than g. In the next section we present one possible way of implementing f, *somewhat efficiently*, using encrypted Bloom filters.

3.2 Implementing f with a Bloom Filter

In Construction 1, we present an encrypted Bloom filter, (m, k)-EBF, with length m and k cryptographic hash functions. Our encrypted Bloom filter differs from a conventional Bloom filter in that the insertion and querying operations are performed homomorphically under encryption. It differs from other encrypted Bloom filter constructions in the literature (*e.g.,* [24, 7, 10]) in that there is no single data holder and the operations are universally verifiable.

While we present (m, k)-EBF as a generic construction, we note it is tailored in certain ways to our needs in the next Section. For example, a sender Alice does not prove she is inserting an element a, or that she even knows an a that satisfies what she is inserting. She only proves that she inserts a single element. The construction also assumes a is a secret when running Insert but is a public value when later running Query.

Encrypted Bloom Filter

An encrypted Bloom filter, (m, k)-EBF, implements the following functions:

- Setup: The authority initializes a length m array EBF with m encryptions of 0 under publicly known randomness: $\langle [\![0]\!], [\![0]\!], \ldots \rangle$. The Authority outputs EBF and a description of k hash functions with output space $[1, m]$.
- $\mathsf{Prepare}(a)$: To verifiably insert a single element into EBF, Alice generates a temporary array EBF_a containing m ciphertexts as follows: Alice then evaluates the set of hash functions on her element a to produce a set of k unique (with high probability) indices. At each of these indices in EBF_a, she inserts $[\![1]\!]$. She inserts $[\![0]\!]$ into the remaining indices and publishes EBF_a. To prove that she inserted only one element without revealing the value a, Alice publishes NIZKPs that prove: (a) each entry in EBF_a is an encryption of 0 or 1; and (b) the homomorphic sum of all entries is an encryption of k. Let the set of proofs be π_a. Alice outputs $\langle \mathsf{EBF}_a, \pi_a \rangle$.
- $\mathsf{Insert}(a, \mathsf{EBF})$: Alice runs $\mathsf{Prepare}(a)$ to obtain $\langle \mathsf{EBF}_a, \pi_a \rangle$. She homomorphically adds the ciphertexts of EBF_a to those of EBF entry-wise, and publishes the resulting array as an updated EBF'. Alice outputs $\langle \mathsf{EBF}, \mathsf{EBF}_a, \mathsf{EBF}', \pi_a \rangle$.
- $\mathsf{Flatten}(\mathsf{EBF})$: Prior to running $\mathsf{Flatten}$, EBF is a counting Bloom filter with an integer between 0 and v at each index, where v is the total number of inserted elements. To convert EBF into a binary Bloom filter, the Authority uses the Mix and Match protocol to generate m blinded tables for the "squashing" function s:

$$s([\![x]\!]) = \begin{cases} [\![0]\!] & x = 0 \\ [\![1]\!] & 1 \leq x \leq v \end{cases}$$

  The Authority evaluates s on each of the m entries in EBF with one of the tables.
- $\mathsf{Query}(a, \mathsf{EBF})$: To test if element a is a member of a binary EBF, the Authority publicly evaluates the set of hash functions on a and retrieves the encrypted entry at each of the k indices. It homomorphically adds these k entries together. Let $[\![x]\!]$ be the result. The Authority the uses the Mix and Match protocol to generate a blinded table for the function f:

$$f([\![x]\!]) = \begin{cases} [\![0]\!] & 0 \leq x \leq k-1 \\ [\![1]\!] & x = k \end{cases}$$

  The Authority evaluates f on $[\![x]\!]$ and publishes the result $[\![t]\!] = f([\![x]\!])$. If the membership test fails, $[\![t]\!]$ contains $[\![0]\!]$ with overwhelming probability and $[\![1]\!]$ otherwise. If the membership test passes, $[\![t]\!] = [\![1]\!]$.

Construction 1: Encrypted Bloom Filter

4 Cobra

In this section we present our proof-of-concept coercion-resistant internet voting scheme Cobra. The high-level phases of an election in the Cobra setting are similar to those in the literature. Unlike the related work, however, numerous pre-computations can be performed prior to the ballot casting phase, and ballot authorization can be performed during the ballot casting phase.

Setup. First the authority runs the distributed key generation DKG outputting a description of $\mathbb{G}_q$, generator $g \in \mathbb{G}_q$, and public key y. They then run encrypted Bloom filter protocol Setup from Construction 1. Additionally, all the Mix & Match tables (except for those in Line 4 of Protocol 3) can be pre-computed at any time during this phase.

Registration (Protocol 1). The first step is an in-person registration phase in which voters create and submit an encrypted credential, which is homomorphically added to the encrypted Bloom Filter. We follow a registration model similar to Selections whereby a voter pre-selects a number of candidate passwords, and registers one of them. The advantage of this approach relative to others in the literature is that the voter does not have to prepare an anti-coercion strategy, *e.g.,* by simulating a zero-knowledge proof. We assume the voter can perform computations, but will discuss how to make it bare-handed in Section 4.3. Registration requires a divertible NIZKP that can be implemented exactly as in Hirt's "K-out-of-L" voting scheme [27]. In fact, EBF entries in our scheme are identical in structure to ballots in Hirt's scheme.

Casting (Protocol 2). To cast a ballot, a voter encrypts their preference (subject to the underlying voting scheme), issuing the appropriate proofs, and using their password to regenerate their credential.

If the voter is being coerced, or wishes to sell a credential, they can simply provide a fake password that they mentally generate at coercion time (*e.g.,* a "panic password" [14]). Real and fake passwords are indistinguishable from the adversary's perspective. The voter can later cast a ballot with their real password, if they have not done so already. Ballots are submitted over an anonymous channel to prevent the adversary from learning if a coerced voter submits (or has submitted) a ballot

Registration

Prior to registration, each eligible voter performs the following steps:

1. The voter chooses a password $\hat{\rho}$ and processes it into credential $\rho = \mathsf{PBKDF}(\hat{\rho}, \mathsf{VoterID})$ with a password-based key derivation function. The voter obtains a description of the Authority's Bloom filter (m, k)-EBF. The voter runs $\mathsf{Prepare}(g^{\rho})$ from Construction 1 on g^{ρ} to generate EBF_a and π_a, which we will rename c and π. Recall c is an array of m ciphertexts with exactly k encryptions of 1 in the indices output by evaluating the k hashes on ρ, and $m - k$ encryptions of 0 in each other entry. Also recall π is a proof that c is well-formed.
2. The voter repeats this process α times, choosing different passwords, to form the set $\{\langle c_1, \pi_1 \rangle, \dots, \langle c_\alpha, \pi_\alpha \rangle\}$. This set will be used in a cut-and-choose protocol with soundness $1 - 1/\alpha$.

Registration is run between each voter and a registrar over an untappable channel:

1. The voter sends their prepared $\{\langle c_1, \pi_1 \rangle, \dots, \langle c_\alpha, \pi_\alpha \rangle\}$ to the registrar. The voter will select one of the α instances, $s \leftarrow [1, \alpha]$, to register and will audit the correct rerandomization of the rest. The voter also sends a commitment to s.
2. For each c_i, the Registrar rerandomizes each ciphertext in the array to produce array $c'_i = \mathsf{ReRand}(c_i)$ and records the random factors in array r'_i. The Registrar prints out $\{\langle c'_1, r'_1 \rangle, \dots, \langle c'_\alpha, r'_\alpha \rangle\}$ and gives it to the voter.
3. The voter reveals s to the registrar and erases $\langle c_s, r'_s \rangle$. The voter checks that $c'_i = \mathsf{ReRand}(c_i, r'_i)$ for all $i \neq s$.
4. If satisfied, the voter with plaintext knowledge of c_s and registrar with knowledge of r'_s generate a joint proof π'_s that c'_s is properly formed.
5. The registrar publishes $\langle \mathsf{VoterID}, c'_s, \pi'_s \rangle$ on the Roster.

After all voters have registered and before voting opens, the Authority does the following:

1. The Authority takes each c_s entry on the Roster and computes, entry-wise, its homomorphic sum. It then runs the $\mathsf{Flatten}$ protocol from Construction 1 on it to create a final encrypted binary Bloom filter EBF that holds a ρ from each voter. Note at this point, any changes to the Roster become non-trivial to make.

Protocol 1: Registration

Ballot Casting

To cast a ballot, a voter performs the following steps:

1. The voter encodes their password (real or panic) $\hat{\rho}$ into credential $\rho = \mathsf{PBKDF}(\hat{\rho}, \mathsf{VoterID})$. The voter commits to their credential, g^{ρ}, and computes a NIZKP of knowledge, π_1, of ρ.
2. The voter encrypts their vote as $[\![v]\!]$ and computes a NIZKP that it is well-formed: π_2. The challenges in both π_1 and π_2 are functionally-dependent on $\langle g^{\rho}, [\![v]\!] \rangle$ using Fiat-Shamir. The voter outputs ballot $\langle g^{\rho}, [\![v]\!], \pi_1, \pi_2 \rangle$.

The election authority posts $\langle g^{\rho}, [\![v]\!], \pi_1, \pi_2 \rangle$ to the public list $\mathsf{AllVotes}$.

Protocol 2: Ballot Casting

with a different password than the one provided to the adversary.

If the voter is changing a previously cast vote, the g^{ρ} value will be the same and the trustees can eliminate the older ballot (the same mechanism used in [5, 6, 15]). Being able to detect duplicate ballots does not violate coercion-resistance, as real and fake ballots will have different values. If a voter is coerced more than once (or sells a vote to each party), they can make up different fake passwords for each interaction.

Ballot Authorization (Protocol 3). As the Authority receives ballots, they perform an authorization step: the encrypted credential is used to perform an encrypted query of the Bloom Filter. If the credential is invalid, *i.e.,* is not contained in the encrypted Bloom filter, the preferences contained inside the encrypted ballot are "zeroed." If the credential is valid, *i.e.,* is contained in the encrypted Bloom filter, the preferences inside the encrypted ballot remain unchanged.

Ballot authorization done concurrently with ballot casting requires trustees to be online and capable of performing computations. This is a departure from other cryptographic voting systems, which generally try to limit the involvement of trustees to after the election. That said, we believe mitigating the impact of board flooding is well worth the trade-off of online work, fur-

Ballot Authorization

An authorized set of trustees can process the ballots as they are posted to the AllVotes list,

1. Upon receiving $\langle g^\rho, [\![v]\!], \pi_1, \pi_2\rangle$, the trustees first check proofs π_1 and π_2. The ballot is discarded if either does not verify. Otherwise $\langle g^\rho, [\![v]\!]\rangle$ is posted to the public list ProvedVotes.
2. The trustees examine ProvedVotes for previously cast ballots with the same g^ρ value. If one is found, it is marked (on all lists) as being a duplicate vote and is disregarded in the tally (Protocol 4).
3. The trustees run $\mathsf{Query}(g^\rho, \mathsf{EBF})$ from Construction 1 and receive $[\![t]\!]$. Recall that $[\![t]\!] = [\![1]\!]$ if $g^\rho \in \mathsf{EBF}$, and $[\![0]\!]$ otherwise.
4. The trustees use the Mix & Match protocol to generate a blinded table for the "zeroing" function z:
$$z([\![t]\!], [\![v]\!]) = \begin{cases} [\![0]\!] & t = 0 \\ [\![v]\!] & t = 1 \end{cases}$$
Recall that Mix & Match ensures the outputs of z are rerandomized prior to its evaluation. The trustees evaluate z on $[\![t]\!]$ and $[\![v]\!]$: $[\![v']\!] = z([\![t]\!], [\![v]\!])$. The trustees publish all associated proofs.
5. The trustees post $[\![v']\!]$ to the public list ValidatedVotes.

Protocol 3: Ballot Authorization

ther noting our setting is open ended as to the degree of online interaction. Trustees have the flexibility to decide among themselves when, if, and for how long to be online. For example, ballot authorization could be performed in batches during times when a sufficient number of trustees are available. It could even serve as a *contingency*, that is, as an optimistic approach in which concurrent ballot authorization (and hence the online component) is only initiated at such time as a board flooding attack is detected.

Tallying (Protocol 4). After the election ends, the Authority takes all of the encrypted, authorized ballots and performs the tally. Without loss of generality, the ballots are homomorphically added, and the sum is verifiably decrypted and published.

4.1 Security Analysis

In this section, we address the security of Cobra. We do not perform a rigorous proof of security, however we assert the properties we claim Cobra holds and sketch an argument for it.

Eligibility Verification. To qualify for eligibility verifiability [34], it should be universally verifiable that (1) each vote in the final tally was cast by a registered voter and (2) there is at most one vote per voter. We claim that Cobra achieves (1) with overwhelming probability if we equate a registered voter with a voter in possession of a registered credential, and we claim (2) holds with the same probability. It is possible that $\mathsf{Query}(g^\rho, \mathsf{EBF})$ returns a false positive, and, therefore, possible for unauthorized ballots to be counted in the final tally.[7] An adversary may attempt to craft a g^ρ that, when hashed, produces a database key that overlaps with the keys of other cast ballots, generating a false positive. Assuming the hashes are cryptographic, there is no non-negligible way for the adversary to find a suitable pre-image better than brute-force. Further, the adversary must start with credential ρ because a suitable g^ρ cannot be submitted without knowledge of ρ. A ρ mapped to g^ρ and queried has a false-positive probability equivalent to querying a random element. Assume that each of the bits of the Bloom filter can be determined by the adversary. This could happen, for example, if all registered voters cast a single ballot with their actual password. The false positive rate of the Bloom filter is thus parameterized to make false positives roughly as hard as finding the decryption key for the election (*e.g.,* on the order of 2^{-80}).

In Cobra, votes submitted with the same credential are apparent from inspection and flagged as duplicates. Given the first property of eligibility verification holds, the second property can be ascertained from inspection of the transcript, ensuring duplicate ballots are disregarded during the tally.

Integrity. We say Cobra has integrity if the final tally is the correct sum of eligible votes in the election. Each step in the election is accompanied by a universally verifiable proof and integrity follows in the same fashion as in related schemes.

[7] An interesting open problem is to design a system that would instead generate false negatives (*i.e.,* a small probability of disenfranchising eligible voters). This would be preferable as there is no incentive for an adversary to purposefully generate such an error.

Tallying

After the last vote has been cast and pre-tally processing has been completed on all ballots, the trustees perform the following steps:

1. The trustees homomorphically sum all $[\![v']\!]_i$ values on ValidatedVotes (except the ones marked as duplicates): $V = \prod [\![v']\!]_i = [\![\sum v'_i]\!]$.
2. The trustees distributively decrypt V, each generating a NIZKP of correct partial decryption. The trustees publish the proofs and the final tally.

Protocol 4: Tallying

Coercion-Resistance. Recall that a system is coercion-resistant if (1) the voter can always realize their voting intent and (2) an adversary cannot distinguish a fake credential from a real credential better than in an ideal system. For simplicity, an ideal system can be thought of as a correct tally. There are potential ways to tell how a voter voted from the tally alone (for example if they are the only voter), and this does not violate the property of coercion-resistance.

Cobra achieves the first property of coercion resistance by allowing voters to never have to release their real credential to an adversary, thus allowing them to use it to cast a ballot of their choosing. Since ballot submission is anonymous (fully, not just an anonymity set as in other systems [15, 44]), this ballot cannot be linked to the voter except by comparing it to the encrypted value on the Roster, which yields no useful information under the CPA-security of the underlying encryption scheme.

A necessary condition for the second property to hold is a coercion-resistant registration process. An adversary that corrupts the voter can demand the voter registers the EBF entry associated with a particular password. In this case, the voter would substitute in their own password and associated EBF entry, and select it from the set of passwords involved in the cut and choose protocol. Since all information about this entry other than the final ciphertexts is erased, an adversary cannot distinguish the final ciphertexts from a rerandomization of the ciphertexts he provided due to the CPA-security of the encryption scheme. Erasures would, of course, need to be enforced, perhaps by having each password on a separate thumb drive and destroying it in front of an official. The commitment could also be physical by having the voter mark which EBF entry they will register (as long as the machine doing the rerandomizations cannot tell which is committed to).

Given a coercion-resistant registration, a voter can always evade the adversary by providing a fake password during ballot casting. The ballot authorization step never reveals, in plaintext, whether a provided credential is real or not.

Coercion-Resistance in Multiple Elections. In its present form, Cobra does not maintain coercion-resistance if a credential is used across multiple elections. To see this, consider an election that is run including a voter Alice, and a set of other voters. After this election, Alice moves to another jurisdiction and a second election is run including Alice and a completely distinct set of voters. Alice must be permitted to submit the same credential g^ρ in both elections in order to achieve the first property of coercion-resistance, and yet this leaves her ballot submission clearly identifiable as, in the worst case, the only common credential between elections (there may be others but we do not want the security of our scheme to rest on the assumption that there must be).

An adversary who coerces Alice in the second election will guess she is cooperating if she can give him a credential that appeared in the first election, and guess she is evading him if she does not. This guess will be correct as long as Alice either did not submit a real ballot in the first election (making the adversary guess she is evading if she cooperates), or submitted a fake credential in the first election (making the adversary guess she is cooperating if she gives the same fake credential). Since we do not *require* either of these, the adversary's guess cannot be made negligible.

4.2 Performance Analysis

We compare the performance of Cobra in Tables 1 and 2 to four other coercion-resistant internet voting schemes, JCJ/Civitas [32, 16], Araujo *et al.* [6], Selections [15], and Schläpfer *et al.* [44]. A rough comparison can be made by counting the number of modular exponentiations. It has the drawback of treating all exponentiations the same[8] and does not account for optimizations due to precomputation or parallelization.

We draw heavily from analysis completed elsewhere [15, 44]. This analysis makes a number of assumptions across all systems to facilitate a better comparison: the use of exponential Elgamal, standard

[8]Except that we do discount the cost of multi-exponentiations.

Registration *(Before Election)*

	JCJ/Civitas	ARRTY	Selections	SHKS	Cobra
Voter	11	9	$(4\alpha - 1)$	11	$(4\alpha + 8)116V + 6$
Registrar	8	10	2α	8	$(2\alpha + 12)116V + 6$

Casting *(During Election)*

	JCJ/Civitas	ARRTY	Selections	SHKS	Cobra
Submit Ballot	$8C + 2$	$8C + 2$	$8C + 2$	$8C + 2$	$8C + 2$
Submit Credential	3	13	$4\beta + 2$	3	2

Processing *(During Election)*

	JCJ/Civitas	ARRTY	Selections	SHKS	Cobra
Check Ballots	$(4C + 4)B$	$(4C + 4)B$	$(4C + 4)B$	$(4C + 4)B$	$(4C + 4)B$
Ballot Authorization	0	0	$(4\beta)B$	0	$(280T + 12CT$ $+19T + 2)B$

Processing & Tallying *(After Election)*

	JCJ/Civitas	ARRTY	Selections	SHKS	Cobra
Ballot Authorization	$(6CT + 7VT + \frac{5}{2}T)B$ $+(\frac{7}{2}T)B^2$	$(24CT + 14T$ $+6)B$	$(12CT + 7T)B$	$(12C\beta T + 7\beta T$ $+7T)B$	0
Tally Ballots	$3TC$	$3TC$	$3TC$	$3TC$	$3TC$

Table 1: Performance comparison in number of modular exponentiations assuming V registered voters, C candidates, B ballots cast and T trustees. α and β are system-specific parameters. Registration and casting cites the work per voter/ballot, while processing and tallying cites the work for all voters and ballots.

Registration *(Before Election)*

	JCJ/Civitas	ARRTY	Selections	SHKS	Cobra
Voter	11	9	39	11	$55,680,006$
Registrar	8	10	20	8	$37,120,006$

Casting *(During Election)*

	JCJ/Civitas	ARRTY	Selections	SHKS	Cobra
Submit Ballot	42	42	42	42	42
Submit Credential	3	13	202	3	2

Processing *(During Election)*

	JCJ/Civitas	ARRTY	Selections	SHKS	Cobra
Check Ballots	$240,000$	$240,000$	$240,000$	$240,000$	$240,000$
Ballot Authorization	0	0	$2,000,000$	0	$10,790,000$

Processing & Tallying *(After Election)*

	JCJ/Civitas	ARRTY	Selections	SHKS	Cobra
Ballot Authorization	$3,000,960,000$	$4,080,000$	$2,010,000$	100,710,000	0
Tally Ballots	45	45	45	45	45

Table 2: Performance comparison in number of modular exponentiations for a moderately-sized election scenario: 5 candidates, 10,000 registered voters, 20,000 submitted ballots, and 3 trustees. Parameters $\alpha = 10$ and $\beta = 50$ follow examples from the literature.

Σ-protocols, and a single registrar. We also note that all of these systems use very similar primitives and protocols, which makes the comparison more compelling.

In contrast to previous comparisons in the literature, we demarcate how much ballot authorization can be done concurrently with ballot casting. We also use a slightly different ballot structure. See Appendix A for further details on our analysis.

Table 1 shows election complexity in terms of modular exponentiations in an election involving V registered voters, C candidates, B cast ballots and T trustees. α is an integer (*e.g.,* 10) that serves as a soundness parameter for registration in Cobra and Selections. β is an integer in $[1, V]$ that serves as an anonymity set in Selections and Shlapfer *et al.* For Cobra, we assume an optimally loaded Bloom filter with false positive probability of 2^{-80}, requiring $m = 116V$ bits, with $k = 80$ hash functions for element insertion and membership testing.

Observe that the distributions of computational work load of the related systems are all similar in that a significant amount of computation occurs after the election closes in the context of ballot authorization. By contrast, Cobra pushes back its computations to the registration period, and allows for concurrently authorizing ballots during the election period. Because Cobra authorizes ballots as they are submitted, the elections results are effectively available immediately after the election ends.

A Sample Election. We provide concrete numbers in Table 2 for a hypothetical election of moderate size involving 5 candidates, 10,000 registered voters, 20,000 submitted ballots, and 3 trustees.

As a rough number, assume a single CPU core can compute 1000 modular exponentiations per second. The voter's part of the registration protocol in our modestly sized election, therefore, would last for almost 2 hours on a fully paralleled 8-core machine. The computational requirements of the registrar, of course, would be considerably more. On the other hand, authorizing a submitted ballot would take on the order of one CPU second, reasonably permitting real-time processing of ballots at submission time.

Since the ballots are authorized prior to the end of the election period, tallying—consisting only of modular multiplications and one verifiable decryption—can be processed almost instantly. By comparison, the fastest related scheme, Selections, would still require over a CPU-hour to produce results after the polls closed. Finally, in the event of a board flooding attack, the wait-time of the related schemes would increase proportionally, whereas Cobra would have the opportunity to amortize this computation over the submission period.

Interpretation of Performance Results. As we have shown, previous proposals for coercion-resistant internet voting are computationally top-heavy: all the number crunching is being done at precisely the time the electorate is eager to learn the results. Cobra's registration protocol is quadratic in the number of registered voters. Though impractical, we have shown it possible for coercion-resistant elections to make results available immediately after polls close.

4.3 Performance Optimizations

Registration requires voters to use a computational device to generate the joint proof that the EBF entry is correctly formed. Instead of having the voter prove it, the trustees could test it. In this case, registration could be bare-handed with preprocessing [42]. To test, the trustees would precompute a set of m tuples $\langle [\![0]\!], [\![1]\!] \rangle$ and verifiably shuffle their order (*e.g.,* with a switching gate [1, 28] or 2-input mixnet). They would test if each entry is a 0 or 1 through a plaintext equality test with the ciphertexts in the tuple. If it matches one of them, the test passes and if it matches neither, the test fails. To test that the EBF contains exactly k encryptions of 1, the entries can be added homomorphically and decrypted.

Instead of using exponential Elgamal, we could consider a pairing-based cryptosystem[9] like BGN for further optimizations [9] . The Freeman variant [22] is implemented in a prime order group which may better allow efficient distributed key generation and distributed decryption. BGN is additively homomorphic (like exponential Elgamal, decryption is possible only in small plaintext spaces) plus it offers a single homomorphic multiplication between two ciphertexts that have not been multiplied before.[10]

With BGN, the test of correctness for the EBF entries can be done very efficiently with a multiplication. For encrypted entry $[\![x]\!]$, the trustees compute $[\![x]\!]([\![x]\!] - 1)$ and decrypt the result. Iff 0, the test passes. Tests on all entries can be batched together by computing $\sum r_i [\![x_i]\!]([\![x_i]\!] - 1) \overset{?}{=} 0$ for randomly chosen constants r_i [9]. Further, step 4 of Protocol 3 can be done without SFE by simply multiplying $[\![v']\!] = [\![t]\!] \cdot [\![v]\!]$.

5 Related Work

The first coercion-resistant remote voting system was proposed by Juels *et al.* [32], which we refer to JCJ. It was slightly refined and implemented as Civitas [16]. Tallying in JCJ/Civitas is expensive: quadratic in the number of submitted ballots. A number of initial attempts at reducing the complexity [2, 47, 46, 38, 50] have been broken [5, 16, 6]. Other attempts have been more successful.

Araujo *et al.* provide a linear-time system [5, 6]. In this system, voter credentials are essentially signed under encryption and during vote submission, the signature can be checked. It has one difficulty: signed values cannot be revoked without a change of keys making it difficult to remove voters from the Roster. Spycher *et al.*'s proposal [49] and Selections [15] both exploit the fact that voters know where they are in the Roster and can include this information in their ballots. The former requires some trust in the election trustees. In the latter, ballot submission is only anonymous within an anonymity set. Anonymity sets are also used by Schläpfer *et al.* [44] who additionally trade-off the amount of work the voter performs in Selections with additional work during ballot authorization.

A few systems offer protection against weaker forms of coercion resistance. Some systems [51, 52] allow a voter to deceive an adversary (Property 2 in the coercion-resistance definition) but doing so prevents them from re-

[9]Note that the security of our system is not impacted by removing the DDH assumption if Elgamal itself is replaced with a CPA-secure scheme in the new setting

[10]The multiplication is achieved via the pairing, so the output ciphertext is in a different group than the inputs. Addition can only occur between pre- or post-multiplication ciphertexts.

liably submitting the ballot of their choice (Property 1). In these systems, the voter registers a blinding factor under encryption, and submits a ballot offset by this factor. If the factors match, the vote is reconstructed but if the voter lies about their factor to an adversary (who does not tell them who he is voting for), it may result in a random vote. Other systems [37, 48] allow voters to cast more than one ballot, overwriting their previous vote. This allows voters to update a coerced ballot but only if they are not coerced, or arrange to sell their votes, at the end of the election. Finally, one system [41] protects against remote adversaries, assuming the coercer does not interact in real-time with the voter during vote casting.

Cryptographic internet voting systems have also been designed for low-coercion elections. These include Helios [3, 4, 11]. Other Internet voting systems concentrate on the untrusted platform issue through "code voting," where a translation of codes to candidates is given to the voter out-of-band [12, 31, 43, 25].

Board flooding attacks are considered by Koenig *et al.* [33]. They mitigate the problem by issuing a finite number of tokens to registered votes, and a token is required to submit a ballot. Voters receive a random number of tokens so they can still deceive a coercer. The details are more complex (the token and credential are combined into a single unit). We consider this approach supplementary to addressing what we feel is the real root of the problem: that ballot authorization is expensive and, prior to this work, could not be done on ballots as they are submitted.

6 Concluding Remarks

In this paper, we introduced the notion of concurrent ballot authorization for coercion-resistant internet voting. With concurrent ballot authorization, ballots can be authorized as they are submitted, allowing near instant reporting of results after the polls close. We also proposed Cobra, a proof-of-concept construction for concurrent ballot authorization.

Although Cobra offers the fastest tally relative to related work, it requires a registration process that is quadratic in the number of eligible voters, making it not viable for practical elections. We hope our first step in this direction will interest researchers in finding new methods that offer concurrent ballot authorization with linear registration (or at least quadratic with a smaller constant). Another interesting open problem is allowing the safe reuse of registered credentials across multiple elections with Cobra.

Acknowledgments

We thank the reviewers for useful and constructive comments. The authors were supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC); the first and second through Postdoctoral Fellowships, and the third by a Discovery Grant.

References

[1] ABE, M. Mix-networks on permutation networks. In *ASIACRYPT* (1999).

[2] ACQUISTI, A. Receipt-free homomorphic elections and write-in ballots. Tech. rep., IACR Eprint Report 2004/105, 2004.

[3] ADIDA, B. Helios: web-based open-audit voting. In *USENIX Security Symposium* (2008), pp. 335–348.

[4] ADIDA, B., MARNEFFE, O. D., PEREIRA, O., AND QUISQUATER, J.-J. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *EVT/WOTE* (2009).

[5] ARAUJO, R., FOULLE, S., AND TRAORÉ, J. A practical and secure coercion-resistant scheme for remote elections. In *Frontiers of Electronic Voting* (2007).

[6] ARAUJO, R., RAJEB, N. B., ROBBANA, R., TRAORÉ, J., AND YOUSFI, S. Towards practical and secure coercion-resistant electronic elections. In *CANS* (2010).

[7] BELLOVIN, S. M., AND CHESWICK, W. R. Privacy-enhanced searches using encrypted Bloom filters. Tech. rep., Columbia, 2007.

[8] BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13*, 7 (1970).

[9] BONEH, D., GOH, E.-J., AND NISSAM, K. Evaluating 2-DNF formulas on ciphertexts. In *TCC* (2005).

[10] BONEH, D., KUSHILEVITZ, E., OSTROVSKY, R., AND SKEITH, W. E. Public key encryption that allows pir queries. In *CRYPTO* (2007).

[11] BULENS, P., GIRY, D., AND PEREIRA, O. Running mixnet-based elections with helios. In *EVT/WOTE* (2011).

[12] CHAUM, D. Surevote: Technical overview. In *WOTE* (2001).

[13] CHAUM, D., AND PEDERSEN, T. P. Wallet databases with observers. In *CRYPTO* (1992).

[14] CLARK, J., AND HENGARTNER, U. Panic passwords: Authenticating under duress. In *USENIX HotSec* (2008).

[15] CLARK, J., AND HENGARTNER, U. Selections: Internet voting with over-the-shoulder coercion-resistance. In *Financial Cryptography* (2011).

[16] CLARKSON, M. R., CHONG, S., AND MYERS, A. C. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy* (2008), pp. 354–368.

[17] CRAMER, R., DAMGÅRD, I., AND SCHOENMAKERS, B. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO* (1994).

[18] CRAMER, R., GENNARO, R., AND SCHOENMAKERS, B. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT* (1997).

[19] CROSBY, S. A., AND WALLACH, D. S. Denial of service via algorithmic complexity attacks. In *USENIX Security Symposium* (2003).

[20] Fiat, A., and Shamir, A. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO* (1986), pp. 186–194.

[21] Freedman, M. J., Nissim, K., and Pinkas, B. Efficient private matching and set intersection. In *EUROCRYPT* (1994).

[22] Freeman, D. M. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *EUROCRYPT* (2010).

[23] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT* (1999).

[24] Goh, E.-J. Secure indexes. Tech. rep., IACR Eprint Report 2003/216, 2003.

[25] Heiberg, S., Lipmaa, H., and Laenen, F. v. On e-vote integrity in the case of malicious voter computers. In *ESORICS* (2010).

[26] Hirt, M. *Multi-Party Computation: Efficient Protocols, General Adversaries and Voting*. PhD thesis, ETH Zurich, 2001.

[27] Hirt, M. Receipt-free K-out-of-L voting based on ElGamal encryption. In *Towards Trustworthy Elections*, vol. 6000 of *LNCS*. Springer, 2010.

[28] Jakobsson, M., and Juels, A. Millimix: Mixing in small batches. Tech. rep., DIMACS, 1999.

[29] Jakobsson, M., and Juels, A. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT* (2000).

[30] Jakobsson, M., Sako, K., and Impagliazzo, R. Designated verifier proofs and their applications. In *EUROCRYPT* (1996).

[31] Joaquim, R., and Ribeiro, C. Codevoting: protection against automatic vote manipulation in an uncontrolled environment. In *VOTE-ID* (2007).

[32] Juels, A., Catalano, D., and Jacobsson, M. Coercion-resistant electronic elections. In *ACM WPES* (2005).

[33] Koenig, R., Haenni, R., and Fischli, S. Preventing board flooding attacks in coercion-resistant electronic voting schemes. In *SEC* (2011).

[34] Kremer, S., Ryan, M., and Smyth, B. Election verifiability in electronic voting protocols. In *ESORICS* (2010).

[35] Küsters, R., and Truderung, T. An epistemic approach to coercion-resistance for electronic voting protocols. In *IEEE Symposium on Security and Privacy* (2009).

[36] Küsters, R., Truderung, T., and Vogt, A. A game-based definition of coercion-resistance and its application. In *CSF* (2010).

[37] Kutylowski, M., and Zagorski, F. Verifiable internet voting solving secure platform problem. In *IWSEC* (2007).

[38] Meng, B. A coercion-resistant internet voting protocol. In *IC-SNC* (2007).

[39] Okamoto, T., and Ohta, K. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In *EUROCRYPT* (1989).

[40] Pedersen, T. P. A threshold cryptosystem without a trusted party. In *EUROCRYPT* (1991).

[41] Raykova, M., and Wagner, D. Verifable remote voting with large scale coercion resistance. Tech. Rep. CUCS-041-11, Columbia, 2011.

[42] Riva, B., and Ta-Shma, A. Bare-handed electronic voting with pre-processing. In *EVT* (2007).

[43] Ryan, P. Y. A., and Teague, V. Pretty good democracy. In *Workshop on Security Protocols* (2009).

[44] Schläpfer, M., Haenni, R., Koenig, R., and Spycher, O. Efficient vote authorization in coercion-resistant internet voting. In *VOTE-ID* (2011).

[45] Schnorr, C. P. Efficient signature generation by smart cards. *Journal of Cryptography 4* (1991).

[46] Schweisgut, J. Coercion-resistant electronic elections with observer. In *EVOTE* (2006).

[47] Smith, W. D. New cryptographic election protocol with best-known theoretical properties. In *Frontiers in Electronic Elections* (2005).

[48] Spycher, O., Haenni, R., and Dubuis, E. Coercion-resistant hybrid voting systems. In *EVOTE* (2010).

[49] Spycher, O., Koenig, R., Haenni, R., and Schläpfer, M. A new approach towards coercion-resistant remote e-voting in linear time. In *Financial Cryptography* (2011).

[50] Weber, S. G., Araujo, R. S. d., and Buchmann, J. On coercion-resistant electronic elections with linear work. In *ARES* (2007).

[51] Wen, R., and Buckland, R. Masked ballot voting for receipt-free online elections. In *VOTE-ID* (2009).

[52] Yi, X., and Okamoto, E. Practical remote end-to-end voting scheme. In *EGOVIS* (2011).

A Further Details on Performance

We assume in all systems that all votes are cast with a standard multi-candidate cryptographic ballot suitable for homomorphic tallying under exponential Elgamal. The voter forms a vector of bit encryptions, one for each candidate, with $[\![1]\!]$ assigned to the candidate they want to vote for and $[\![0]\!]$ for the rest (as per [26] and many other voting schemes).

Even though ballots are mixed in JCJ, Civitas, Araujo *et al.*, Selections, and Shläpfer *et al.*, and each ballot can be simply decrypted, it is cheaper to decrypt their homomorphic sum than each ballot individually (1 distributed decryption vs. B). However the cost to the voter is slightly higher as they must prove the ballot is well-formed. Recall though, for coercion-resistance, the voter still has to prove their ballot encrypts a correct candidate in mix-based tallying (otherwise an adversary wanting to test if the voter supplied the correct credential would vote for a random string and see if it shows up in the final decryption stage—an adaptation of a pattern/Italian attack). Given the voter must do a proof linear in the number of candidates either way, the work for the voter does not increase much with the change ($8C+2$ modular exponentiations vs. $4C+2$).

We use the number of modular exponentiations for common primitives according to Table 1 in [44]. For Cobra, registration requires the voter to do αm encryptions ($2\alpha m$). The registrar then rerandomizes each encryption ($2\alpha m$) and provides the random factors. For the $\alpha - 1$

EBF entries the voter audits, the voter checks their rerandomization $(2m(\alpha-1))$. For the selected entry, the voter and registrar do a joint proof that each ciphertext is a 0 or 1 and that they sum to k. The voter proves each is 0 or 1 $(6m)$ and proves they sum to k: 0 to sum and 2 to prove decryption. The registrar $(8m)$ simulates the same proofs $(8m$ and 2) and generates the diverted proof. Both should verify the final diverted proofs $(4m$ and $4)$.

To cast a ballot, the voter constructs a vote using a vector of encrypted bits equal to the length of the candidate $(2C)$ and proves each is a 0 or 1 $(6C)$. They also prove the sum decrypts to 1 (2). For authorization, the voter commits to their credential (1) and proves knowledge of it (1).

Upon receiving a ballot, the authority checks that the candidate vector is properly formed $(4C+4)$. They check the proof of knowledge of the commitment (2). They query the BF by hashing the credential, fetching the entries and summing them (free). They then conduct a mix and match, which consists of k plaintext equality tests (PETs) at worst and $k/2$ on average (we assume tables are precomputed during setup). Performing the plaintext equality test is $(k/2)(3T)$ and proving correctness is $(k/2)(4T)$. Next, the trustees perform the second mix and match. Here the table must be generated since $[\![v]\!]$ is not known before the election. Mixing 2 tuples of length $C+1$ (input plus a ciphertext for each candidate) is $8(C+1)T$ and the generating proofs is $4(C+1)T$. Evaluating the SFE is a single PET on average: $3T$ to do and $4T$ to prove.