Civitas: A Secure Remote Voting System

Michael R. Clarkson   Stephen Chong   Andrew C. Myers
Department of Computer Science
Cornell University
{clarkson,schong,andru}@cs.cornell.edu

Cornell University Computing and Information Science
Technical Report TR2007-2081
May, 2007
Updated August, 2007*

Abstract

Civitas is the first implementation of a coercion-resistant, universally verifiable, remote voting scheme. This paper describes the design of Civitas, details the cryptographic protocols used in its construction, and illustrates how language-enforced information-flow security policies yield assurance in the implementation. The performance of Civitas scales well in the number of voters and offers reasonable tradeoffs between time, cost, and security. These results suggest that secure electronic voting is achievable.

1 Introduction

Electronic voting is now a reality—and so are the many errors and vulnerabilities in commercial voting systems [50, 3, 10]. Voting systems are hard to make trustworthy because they have strong, conflicting security requirements. Confidentiality of votes must be maintained to protect voters' privacy, to prevent selling of votes, and to defend voters against coercion. Integrity of election results must be assured so that all voters are convinced that votes are counted correctly. And, if an election authority attempts to corrupt the integrity of an election, the attempt must be detected and attributed correctly. The goals of confidentiality and integrity are in tension: a public show of hands guarantees integrity but destroys confidentiality, whereas secret ballots typically guarantee confidentiality but fail (due to possible miscounts, ballot stuffing, etc.) to assure integrity. Availability of the voting system during elections is also crucial to prevent disenfranchisement of voters. Given the civic importance of elections, the consequences of violating these requirements can be dramatic.

*The original name of this voting system was *CIVS*. This update changes the name to *Civitas*.

Dagstuhl Seminar Proceedings 07311
Frontiers of Electronic Voting
http://drops.dagstuhl.de/opus/volltexte/2008/1296

Attention on electronic voting has focused on the problem of *supervised voting*, in which voters interact with a computing device under the supervision of election authorities. However, we argue that the right problem to solve is the more general problem of *remote voting*, in which there is no supervision of voters as they cast their votes. Internet voting, an instance of remote voting, is increasingly used by diverse and important groups, such as open-source software communities, e.g., Debian [24]; the Association for Computing Machinery [2]; and, though it ultimately canceled the project, the U.S. military [44]. Even U.S. government elections appear to be reducing the supervision of voting. Voters in the state of Oregon vote by mail, not at polling places, and all states receive a substantial fraction—enough to change the outcome of many elections—of their ballots by mail as absentee ballots.

Many security experts have been skeptical about electronic voting [64, 26, 29, 45, 56], arguing that the risks of deploying electronic voting are too great and that it is too hard to convince voters of the security of election mechanisms. Yet as hard as secure supervised voting may be, secure remote voting is even harder. Electronic voting systems are often designed for trusted polling places with human supervision and certified hardware and software. But when voters can vote from any Internet host, election servers can be controlled by many parties, and communications can be tapped and modified, every component of a remote system is potentially malicious.

Despite existing pessimism about secure voting, this paper argues that it is possible today to build a secure, practical, remote voting system. Civitas is our prototype. It is constructed using principled techniques. First, the design is a refinement of a cryptographic voting scheme proven secure by Juels, Catalano, and Jakobsson [47]. No previously implemented voting system has implemented this voting scheme or any other voting scheme that offers comparable security. Second, to increase assurance in the voting software, the implementation was carried out in Jif [57, 59], a language whose compiler and run-time system enforce information-flow security policies. Civitas also implements sophisticated preferential voting methods in which voters may rank candidates. Such methods are resistant to strategic voting, yet care must be taken to ensure they do not create a covert channel that violates confidentiality. Note the distinction between voting *systems*, which are software implementations, voting *schemes*, which are cryptographic protocols, and voting *methods*, which are algorithms for choosing between candidates.

Previous work on secure voting systems falls roughly into two categories: theoretical voting schemes that have not been implemented and have not been shown to scale to large elections, and voting system implementations that do appear to scale to large elections but offer insufficient assurance of important confidentiality and integrity requirements. Prior work therefore illustrates a conflict between security and practicality. Civitas resolves this conflict by demonstrating for the first time that strong security properties can be offered by a practical remote voting system.

The development of Civitas has led to several contributions:

- The first voting system which implements a voting scheme that provably satisfies strong security properties.

- A provably secure voter registration protocol, which distributes trust over a collection of registration authorities.
- A scalable design for vote storage that ensures election integrity without expensive fault tolerance mechanisms.
- A performance study demonstrating the practicality of secure remote voting.
- A concrete, publicly available specification of the cryptographic protocols required for a remote voting scheme.

This paper describes and evaluates Civitas. Section 2 explains the security requirements and threat model for a coercion-resistant, universally verifiable, remote voting system. The following sections present Civitas: its architecture and voting scheme (Section 3), the trust assumptions on which the scheme is based (Section 4), the implementation of cryptographic components (Section 5), techniques used to achieve scalability (Section 6), the use of Jif to increase assurance in the system (Section 7), and finally enhancements such as preferential voting methods (Section 8). The performance study in Section 9 demonstrates that Civitas is practical for real-world elections, scaling with reasonable tradeoffs between time, cost, and security. Section 10 discusses possible attacks on Civitas and corresponding defenses. Related work is reviewed in Section 11, and Section 12 concludes.

2 Security Model

A *secure* remote voting system must satisfy two strong requirements. The first, a confidentiality requirement, is that no adversary can learn any more about votes than is revealed by the final tally. The second, an integrity requirement, is that no adversary can change the final tally to be different than if all votes were publicly announced and tallied in the presence of all voters. (These intuitive requirements are instances of the privacy and correctness requirements of secure multi-party computation [35].)

Previous work has proposed several security properties to satisfy these requirements. For confidentiality, a common, simple property is *anonymity*: the adversary cannot learn the map from voters to votes. This property is accompanied by the strong assumption that voters will not act to violate their own anonymity. In remote voting, this assumption is unreasonable: vote buying, a well-known phenomenon in real-world supervised elections, could have a large impact by inducing voters to reveal their votes. Moreover, in remote voting, adversaries may be able to coerce voters with virtual, or even physical, threats. Against such adversaries, it is necessary to ensure that voters can convincingly lie to the adversary about the votes they submit. The adversary then cannot trust any claims that voters makes about their votes, guaranteeing that confidentiality is maintained even when voters collude with the adversary. This is a strong property that Civitas is required to satisfy:

Coercion Resistance. *Voters cannot prove whether or how they voted, even if they can interact with the adversary while voting.*

Removing the ability to interact with the adversary results in a weaker property known as *receipt-freeness*, originally defined by Benaloh [6].

"Coercion resistance" is a term used throughout the voting literature, though rarely with a precise, consistent meaning. Recently, Juels et al. [47] and Delaune et al. [25] gave rigorous formal definitions in the computational and symbolic models, respectively, of cryptography. Although the informal definition given above is consistent with both, the rest of this paper uses the term in the same formal sense as Juels et al.

Civitas is also required to satisfy a strong integrity property, originally stated by Sako and Kilian [66]:

Universal Verifiability. *All voters are convinced the final tally is correct. This entails that all votes that were cast are counted, only authorized votes are counted, and no votes are changed during the tabulation of votes.*

It is realistic to assume that the adversary may control some of the voting system, including the network, the machines used to collect and tabulate votes, and the agents providing this infrastructure. Therefore, to offer strong security assurance, we assume that trust can be distributed over a set of *election authorities*, agents who provide the various system components. The system should remain secure as long as a set of election authorities remains uncompromised by the adversary. Section 4 identifies a set that is sufficient to guarantee security of Civitas.

Completing our characterization of the adversary, voting systems should be secure with respect to the following threat model, essentially due to Juels et al. [47]:

Strong Adversary. *The adversary may:*

1. *Corrupt a threshold of the election authorities.*

2. *Coerce voters, demand their secrets, and demand any behavior of them. This may be done remotely or in the physical presence of the voter.*

3. *Control the network by reading, sending, delaying, or dropping any messages sent on public channels.*

4. *Perform any polynomial-time computation.*

The third point in this definition suggests the existence of non-public channels. In fact, we assume the existence of some *anonymous* channels, on which the adversary cannot identify the sender, and some *untappable* channels, which the adversary cannot use at all. An untappable channel must provide perfect secrecy, either by being physically untappable or implementing information-theoretic encryption, e.g., a one-time pad. We argue that these assumptions are reasonable and necessary in Section 4.

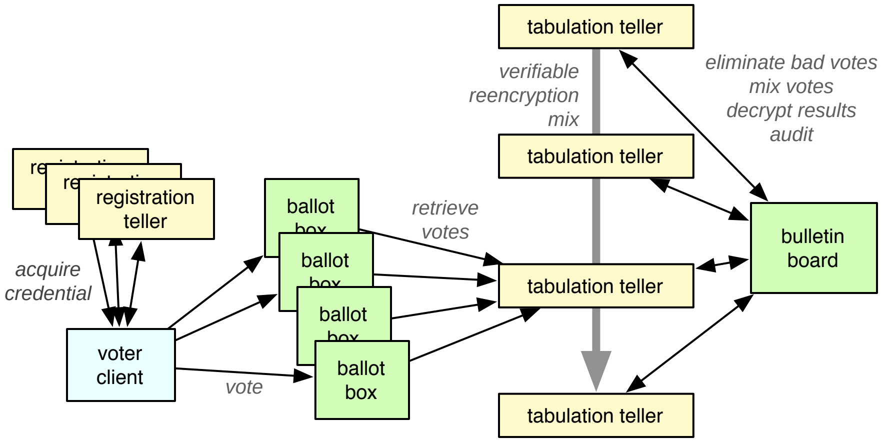

Figure 1: Civitas architecture

3 Architecture

Civitas comprises five (groups of) agents: a supervisor, a registrar, voters, registration tellers, and tabulation tellers. Some of these are depicted in Figure 1. The agents other voters are the election authorities.

- The *supervisor* administers an election. This includes creating the election by specifying the ballot design, the registrar, and the tellers; and starting and stopping the election.

- The *registrar* authorizes voters for an election by posting each voter's identifier (an arbitrary string, perhaps a name or email address) and public key.

- *Registration tellers* are responsible for generating the credentials that voters use to cast their votes.

- *Tabulation tellers* tally votes in a way that is both coercion-resistant and universally verifiable.

These agents carry out the election using two underlying storage services: a publicly readable, insert-only *bulletin board* to which election authorities post messages (which must be signed), and a set of *ballot boxes* that accept votes from voters.

The Civitas voting scheme refines a voting scheme developed by Juels, Catalano, and Jakobsson (JCJ) [47]. Similar to a JCJ election, a Civitas election has three phases: setup, voting, and tabulation. Each election phase involves cryptographic protocols; for simplicity, the discussion of these is deferred to Section 5.

3.1 Setup phase

First, the supervisor creates the election by posting the registrar's public key and the ballot design on an empty bulletin board. The supervisor also identifies the tellers by posting their individual public keys. The agreement of the registrar and the tellers to participate in the election is obtained out-of-band. The registrar posts identifiers for all authorized voters along with their public keys.

Second, the tabulation tellers collectively generate a public key for a distributed public-key cryptosystem, and post it on the bulletin board. Decryption of messages encrypted under the public key requires the participation of all tabulation tellers.

Finally, the registration tellers generate *credentials*, which are used to authenticate votes anonymously. Each credential is associated with a single voter. Like keys in a public-key cryptosystem, credentials are pairs of a public value and a private value: the *public credential* and the *private credential*. All public credentials are posted on the bulletin board. Credentials are generated in a distributed fashion, much like the tabulation teller's public key, such that each registration teller holds a share of the private credentials. Thus, valid private credentials can be forged or leaked only if all registration tellers collude.

3.2 Voting phase

Voters register to acquire their private credential. Each registration teller authenticates a voter using the voter's public key, then releases the registration teller's stored share of the private credential to the voter. The voter combines all of these shares to construct the full private credential.

To vote, the voter submits a private credential and a *choice* of a candidate (both encrypted), along with a proof that the vote is well-formed, to some or all of the ballot boxes. This replication of the vote is used to guarantee availability of the vote for tabulation.

3.3 Tabulation phase

The tabulation tellers collectively tally the election. The operations performed during tabulation are made publicly auditable by posting proofs of correctness. All the tabulation tellers verify these proofs as tabulation proceeds through several subphases.

Retrieve data. All tabulation tellers retrieve the votes from each ballot box and the public credentials from the bulletin board.

Verify proofs. Each vote is checked to verify the proof that the vote is well-formed. Any vote with an invalid proof is discarded.

Eliminate duplicates. Only one submitted vote is retained for each credential. Any duplicates are eliminated according to a revoting policy (see Section 8.2).

Anonymize. Both the list of submitted votes and the list of authorized credentials are anonymized by applying a random permutation, implemented with a *verifiable reencryption mix*. In the mix, each tabulation teller in turn applies its own random permutation.

Eliminate unauthorized votes. The credentials in the anonymized votes are compared against the anonymized authorized credentials. Any votes with invalid credentials are discarded.

Decrypt. The remaining choices, but not credentials, are decrypted. The final tally of the election is publicly computable.

3.4 Lying to an adversary

To lie about a private credential, the voter locally runs an algorithm to produce a fake private credential. This generates fake private credential shares that, to an adversary, are indistinguishable from valid shares. The voter can substitute the fake private credential for his real private credential and behave however the adversary demands. For example, the voter might abstain from voting with the fake credential, submit an adversary-supplied vote with the fake credential, or surrender the fake credential to the adversary.

4 Trust Assumptions

The security of the Civitas voting scheme rests on a number of assumptions.

Trust Assumption 1. *The adversary cannot simulate a voter during registration.*

This assumption helps to guarantee the confidentiality and integrity of the voter's vote. If the adversary could simulate the voter for the entire election, it would be impossible to achieve a secure election. So there must be some period of time during which the adversary cannot simulate the voter. Registration is a good time for this because it requires authentication. Civitas currently accomplishes authentication with something the voter knows: the voter's private key. Although the adversary is allowed to demand voters' secrets, additional mechanisms, such as tamper-resistant hardware, could enforce non-transferability of the voter's private key.

Trust Assumption 2. *Each voter trusts at least one registration teller, and the channel from the voter to the voter's trusted registration teller is untappable.*

This assumption guarantees that the voter can successfully lie about a credential share and construct a convincing fake credential. Faking a credential requires modifying at least one of the shares received by a voter during registration, so at least one registration teller is able to distinguish a fake credential. Moreover, any agent who observed the original share in transit on the network can detect that it was modified, even if the share was encrypted. So an untappable channel is required for distribution of this one share. Other messages in the system can still use public channels that the adversary controls.

While an untappable channel may be a strong trust assumption, it is actually the weakest known assumption for a coercion-resistant voting system [66, 21, 38, 4, 47]. Eliminating this assumption requires a distributed, incoercible construction of credentials. While results on incoercible multi-party computation [13] might help, no practical construction is yet known. This has been an open problem for at least a decade [22].

Trust Assumption 3. *The channels on which voters cast their votes are anonymous.*

This assumption guarantees that the adversary cannot trivially violate coercion resistance by observing network traffic and learning which voters have voted. Even if this assumption fails, an adversary cannot learn the voter's vote or credential. Anonymizing networks [27] could be used to implement an anonymous channel. (Timing attacks on the anonymizing network are unlikely to be effective because a vote typically fits into just three packets.)

Trust Assumption 4. *Voters trust that at least one of the ballot boxes to which they submit their votes is correct.*

A *correct* ballot box returns all the votes that it accepted to all the tabulation tellers. This assumption guarantees that all submitted votes are included in the tabulation process. If instead no ballot boxes were assumed to be correct, voters would need to be able to complain anonymously to the supervisor when their votes are not included in tabulation. But it is simpler and reasonable to assume that at least one ballot box is correct. This is weaker than standard assumptions such as Byzantine fault tolerance [14] (which would require that no more than f ballot boxes fail, for some number f) and multi-party computation [35] (which would require a majority of honest ballot boxes).

Trust Assumption 5. *There exists at least one honest tabulation teller.*

This assumption guarantees confidentiality of votes by ensuring that at least one tabulation teller performs a random permutation during mixing, and that at least one share of the distributed private key is not revealed to the adversary. Without this, the adversary could trivially decrypt all credentials and votes. Additional fault tolerance techniques [31, 18, 67] could make it more difficult for the adversary to corrupt all of the tellers. Note that this assumption is not necessary for integrity, which is enforced through auditing.

Trust Assumption 6. *The Decision Diffie-Hellman (DDH) assumption, the RSA assumption, and that SHA-512 implements a random oracle.*

DDH and RSA are standard cryptographic assumptions. A violation of these yields attacks against many systems—not just Civitas. The more fundamental assumption for Civitas is DDH, as the JCJ security proof is a reduction from it. SHA-512 is discussed in the next section.

5 Cryptographic Components

Implementation of Civitas requires a number of cryptographic components, including a distributed encryption scheme, a mix network, a registration protocol, and a ballot box system. This section gives an overview of how these components are used during an election; Appendix B contains a fuller description of the protocols involved.

Several of the components described below instantiate oracles assumed by the JCJ voting scheme. The security of Civitas follows from the JCJ security proof [47] and the individual security proofs of each component, cited below. For the registration oracle, which we constructed ourselves, we give a security proof in Appendix A.

Many of the components below require posting messages to the bulletin board. All such posts must be signed by the principal who makes the post. Also, a variety of zero-knowledge proofs are used to enforce the honest execution of protocols. These zero-knowledge proofs are all made non-interactive via the Fiat-Shamir heuristic [30], so their security is in the random oracle model [5]. Civitas implements a random oracle with SHA-512.

5.1 Setup phase

Keys. During setup, the supervisor and the registrar both post public keys representing various principals. All election authorities are represented by RSA public keys, whereas voters are represented by El Gamal public keys. The RSA keys are a convenience, since many real-world organizations already have RSA keys, but could be easily replaced by another cryptosystem. The El Gamal keys, however, are necessary for the protocols described below.

Civitas assumes that the supervisor (or registrar) is trusted to certify this binding between public keys and principals, or else that there is some external mechanism to certify the binding. For example, public keys might be certified by a public-key infrastructure, or they might be compiled into the voting software.

Encryption Scheme. Civitas implements a distributed El Gamal scheme as described by Brandt [9]. The supervisor posts a message (p, q, g) describing the cryptosystem parameters: a safe prime $p = 2q + 1$, where q is also prime, and a generator g of the group of quadratic residues $QR_p = \{x^2 \bmod p \mid x \in \mathbb{Z}_p^*\}$. The message space $\mathcal{M}$ for the cryptosystem is the group QR_p. The tabulation tellers run a protocol to generate a public key K_{TT} for which each teller holds a share of the corresponding private key. Encryption of message $m \in \mathcal{M}$ under key K is denoted $\mathsf{Enc}(m; K)$, or simply $\mathsf{Enc}(m)$. Decryption of ciphertext c, denoted (with an implicit private key) as $\mathsf{Dec}(c)$, requires the participation of all tabulation tellers.

El Gamal encryption is *homomorphic* with respect to multiplication. That is, $\mathsf{Enc}(m) \cdot \mathsf{Enc}(n) = \mathsf{Enc}(m \cdot n)$. Encryption is also probabilistic, and it permits a *reencryption* operation, denoted $\mathsf{Reenc}(c)$ for a ciphertext c. Reencryption (which is also probabilistic) produces a new encryption of the same plaintext such that $c \notin \mathsf{Reenc}(c)$. Encryption can also be made *non-malleable*, preventing the use of homomorphisms and reencryption, by the use of Schnorr signatures [69]. Civitas uses non-malleable encryption until the tabulation phase, where malleability is required.

Civitas uses two zero-knowledge proofs to ensure the honesty of tellers during key generation and during decryption. The first is a *proof of knowledge of a discrete logarithm* due to Schnorr [68]. Given a message v and generator g, this proof shows knowledge of an x such that $v \equiv g^x \pmod{p}$. The second is a *proof of equality of discrete logarithms* due to Chaum and Pedersen [16]. Given messages v and w and generators g and h, this proof shows there exists an x such that $v \equiv g^x \pmod{p}$ and $w \equiv h^x \pmod{p}$.

Credential generation. Civitas uses a novel, distributed construction for credential generation and distribution. (Some of the ideas behind this construction are present in earlier work [22, 38, 47].) The security of this construction is proved in Appendix A.

For each voter, each registration teller individually generates a share of a credential, then posts to the bulletin board the public part of the credential share, and stores the private part. The voter's public credential is publicly computable from the posted shares.

More concretely, for each voter, registration teller i, denoted RT_i, chooses a random element $s_i \in \mathcal{M}$. This is the private credential share. RT_i also posts $S_i = \mathsf{Enc}(s_i; K_{\text{TT}})$ on the bulletin board as the public credential share. The resulting public credential S for the voter is $S \triangleq \prod_i \mathsf{Enc}(s_i; K_{\text{TT}}) = \mathsf{Enc}(\prod_i s_i; K_{\text{TT}})$, where the equality follows from the homomorphic property of El Gamal.

5.2 Voting phase

Registration. To acquire a private credential, a voter contacts each registration teller and authenticates using the El Gamal public key posted by the registrar. Civitas uses the Needham-Schroeder-Lowe [53] protocol to authenticate and establish a shared AES session key. The voter requests the registration teller's share s_i of the private credential. The registration teller responds with this share, accompanied by a *designated-verifier reencryption proof* (DVRP) due to Hirt and Sako [38]. This proof convinces the voter—and only the voter—that the private share is correct with respect to the public share posted on the bulletin board, i.e., that S_i is an encryption of s_i. After retrieving all the shares, the voter constructs the private credential s, where $s \triangleq \prod_i s_i$.

Voting. To cast a vote, the voter posts an unsigned message

$$\mathsf{Enc}(s; K_{\text{TT}}), \mathsf{Enc}(v; K_{\text{TT}}), P$$

to the ballot boxes, where s is the voter's private credential, v is the voter's desired choice, and P is a zero-knowledge proof. Proof P shows that the vote is well-formed with respect to the ballot design of the election with a *1-out-of-L reencryption proof* due to Hirt and Sako [38], which given $C = \{c_i \mid 1 \leq i \leq L\}$ and c, shows there exists an i such that $c_i = \mathsf{Reenc}(c)$. Proof P also shows that the submitter knows the plaintext credential and vote, which makes votes non-malleable. This defends against an adversary who attempts to post functions of previously cast votes. Civitas implements this part of P with an adaptation of a proof due to Camenisch and Stadler [11].

Lying to the adversary. To construct a fake credential, the voter chooses at least one registration teller RT_i and lies about the share s_i that registration teller sent to the voter by substituting a new, random group element $s_i' \in \mathcal{M}$. The voter is able to construct a DVRP for this fake share that causes the share to look valid to the adversary, unless the adversary has corrupted the registration teller the voter chose (in which case the adversary already knows the real share), or the adversary observed the channel used by the registration teller and voter during registration (in which case the adversary has seen the real proof). But by Trust Assumption 2, there exists some teller and channel that the adversary does not control, so it is always possible for the voter to lie.

5.3 Tabulation phase

Ballot boxes. When the supervisor closes the election, each ballot box cryptographically commits to its contents and posts a signature of this commitment on the bulletin board. The supervisor then posts his own signature on all these received commitments. This defines the set of votes to be tabulated. Thus, if a voter posts his vote to at least one correct ballot box, then his vote will be included in tabulation. (A malicious supervisor could violate this by excluding a correct ballot box. This trust in the supervisor could be eliminated by running a more expensive agreement protocol.)

An important security property emerges from this construction: the system achieves just enough availability to guarantee universal verifiability, which requires that all votes are available for tabulation. By Trust Assumption 4, at least one ballot box is correct, so this construction achieves availability of votes. The construction does not achieve the complementary property of availability of election results, but see Section 10 for discussion on how to do so. The benefit of this construction is scalability: no heavyweight fault tolerance protocols are needed.

Mix network. A mix network is used to anonymize the submitted votes and authorized credentials. Civitas implements a reencryption mix network made verifiable by randomized partial checking [42]. Each teller in the network performs two permutations, as in Prêt à Voter [17]. Thus, anonymity depends on at least one teller being honest. The probability that an adversary, who controls all but one teller, can violate the anonymity of the mix (by identifying the output corresponding to a given input) is $2/N$, where N is the number of votes in the mix. This gives the adversary an advantage of $1/N$ over simply guessing the correct output. A mix network based on zero-knowledge proofs [33, 60] would offer greater anonymity at the cost of more expensive verification.

Duplicate and invalid vote elimination. It would be easy to eliminate votes containing duplicate or invalid credentials if credentials could be compared by decrypting them. However, this would fail to be coercion-resistant: all valid private credentials would be revealed, so voters could never lie about credentials. Instead, a zero-knowledge protocol called a *plaintext equivalence test* (PET) is used to compare ciphertexts. Given c and c', a PET reveals whether $\text{Dec}(c) = \text{Dec}(c')$, but nothing more about the plaintexts of c and c'. Civitas implements a PET protocol due to Jakobsson and Juels [41].

For duplicate elimination, each anonymized submitted vote must be compared against all others, and for invalid credential elimination, each submitted vote's credential must be

Table 1: Election parameters

V	Number of voters
A	Number of authorities
C	Number of choices on ballot
N	Number of submitted votes
K	Min. number of voters per block
M	Max. number of submitted votes per block

compared against all the anonymized authorized credentials. Because this quadratic time performance would limit scalability, Civitas partitions the set of submitted votes into blocks, as discussed in the next section.

6 Scalability

A practical voting system should scale to a large number of voters, using computing resources proportional to the number of voters. There are two main challenges for scalability in Civitas. First, elimination of duplicate and invalid votes takes quadratic time. Second, tabulation requires each teller to perform computation for each vote.

The solution to both challenges is to assign voters into *blocks*. Voters are guaranteed anonymity within their own block, and the tally for each block can be computed independently from all other blocks. The implementation of blocking is straightforward, requiring that the registrar assigns each voter to a block; each submitted vote identifies, in plaintext, the block in which its (purported) credential resides; and that the vote proof is extended to make this block identifier non-malleable.

The anonymity provided by blocks is similar to that obtained in real-world voting precincts, where a voter is anonymous within their precinct, but precinct results are public. However, assignment into blocks need not be made based on physical location, as precinct assignments are. Registrars can implement any policy on block assignment, but enforcement of this is outside the scope of Civitas. One potential policy is to assign voters to blocks in a way that is verifiably pseudorandom; this may reduce the risk of reprisal by the adversary against an entire block of voters.

To understand the impact of blocking, Table 1 identifies the parameters in which an election scales. Parameter A describes the number of election authorities of each type. For example, if $A = 4$, then there are four registration tellers, four tabulation tellers, and four ballot boxes. Regardless of the value of A, there is a single bulletin board. In terms of these parameters, blocking reduces duplicate elimination from $O(N^2)$ plaintext equivalence tests to $O(BM^2)$, where $B = \lfloor \frac{V}{K} \rfloor$ is the number of blocks. Invalid credential elimination is reduced from $O(VN)$ to $O(BKM)$.

The critical scalability gain from blocking is that the B factor in each of these terms is

Table 2: Modular exponentiations

Agent	Action	Protocol	BB
RT	Gen. all creds.	$4K$	K
	Dist. all creds.	$14K$	–
Voter	Retrieve a cred.	$12A$	A
	Vote	$4C+7$	–
TT	Retrieve	–	$AK+A+1$
	Check proofs	$4M(C+1)$	–
	Dup. elim.	$\binom{M}{2}(8A-1)$	$3A$
	Mixes	$2(A+1)(M+K)$	$2A$
	Invalid elim.	$KM(8A-1)$	$3A$
	Decrypt	$K(4A-1)$	A

easily parallelizable: a different set of machines can be used to implement the tabulation tellers for each block. Given this parallelization, the tabulation time does not depend on the number of voters, since M and K are independent of V. This allows the performance of Civitas to scale independently of the number of voters.

Tabulation time is dominated by the time required to perform modular exponentiations. Table 2 gives the number of modular exponentiations performed by individual agents (each registration teller (RT), tabulation teller (TT), and voter), per block. The table distinguishes *protocol exponentiations*, which are required by the Civitas voting scheme regardless of the implementation of the bulletin board, from *bulletin board (BB) exponentiations*, which are required by the particular implementation used in our prototype. These BB exponentiations result from RSA signatures and verifications. Exponentiations are counted under the conservative assumption that there are no duplicate votes and that no voters abstain.

7 Assurance from Jif

The prototype of Civitas is implemented in an extended version of the Jif 3.0 programming language [57, 59]. Jif is a *security-typed* language [74] in which program variables are annotated with security policies. The Jif compiler verifies that the program uses information in accordance with these policies. This specification and automatic enforcement of security policies provides assurance that our implementation satisfies the security requirements of Civitas.

In Jif, security policies are expressed with *decentralized security labels* [58], which allow specification of confidentiality and integrity requirements of principals. For example, the confidentiality label $\{\texttt{RT}_1 \rightarrow \texttt{alice}\}$ means that principal $\texttt{RT}_1$ owns the labeled information but permits principal alice to read it. Similarly, the integrity label $\{\texttt{RT}_1 \leftarrow \texttt{supervisor}\}$ means that $\texttt{RT}_1$ permits principal supervisor to change the labeled information. Because labels express security requirements explicitly in terms of principals, they are useful for systems, like Civitas, where principals need to cooperate yet are mutually distrusting.

The Jif extension in which Civitas is implemented adds *declassification until* and *erasure* confidentiality policies [19]. These allow principals to state when the set of readers of information may be expanded, thus declassifying information, and when the set of readers must become more restrictive, thus ensuring erasure of information. These kinds of policies are useful in expressing security requirements in the implementation of Civitas.

For example, in the implementation of the mix networks, each tabulation teller must commit to random bits. After all tabulation tellers have committed, the random bits are revealed, combined, and used to audit the mix. The security of auditing relies on the random bits not being revealed until all tabulation tellers have committed. The confidentiality label for the random bits of $\mathtt{TT}_i$ indicates that the information is readable only by $\mathtt{TT}_i$ until a condition *allCommitted* is satisfied, whereupon the information may be declassified to be readable by all principals. The condition *allCommitted* is set programmatically at the program point where all commitments from other tellers have been received.

Erasure policies are used during the registration of voters with the registration tellers. Each registration teller creates a private credential share for each voter, and must store this share until the voter requests it. After the voter receives this share from the registration teller, the teller should erase it. This ensures that even if the teller is subsequently compromised, the voter's private credential will not be revealed. The erasure policy for the teller's copy of the share indicates that when the share has been delivered to the voter, the credential becomes unreadable by any principal. The Jif compiler inserts code to overwrite the information when the condition *delivered* becomes satisfied.

The Civitas prototype implementation totals about 14,000 lines of Jif code. An additional 8,000 lines of Java code are used to perform socket and file input and output, and to implement cryptographic operations (i.e., El Gamal and zero-knowledge proofs). Cryptography was implemented in Java rather than Jif because accurate modeling of cryptography in the decentralized label model is an open problem [37, 1].

8 Enhancements

8.1 Preferential voting methods

In a preferential election method, each voter submits a (partial or total) ordering of the candidates according to his preference. Examples of such methods include the widely used Single Transferable Vote (aka Instant-Runoff Voting) method, the Condorcet method, and the Borda count [8]. Such methods are attractive because they allow voters to express detailed information about their preferences and because they are resistant to manipulation by strategic voting.

No electronic voting systems of which we are aware have considered coercion-resistant implementation of a preferential method. A difficulty with preferential methods is that they introduce a covert channel: voters can encode information into their vote by changing lower-order preferences. For example, if there are twenty candidates, a voter's lowest ten preferences probably will not influence the outcome of the election, so at least $10!$ distinct values can be covertly encoded into the vote. Since Civitas publicly posts votes, voters can be coerced into encoding their identity into their votes.

Civitas implements a Condorcet method, in addition to more traditional ballots. Condorcet methods enforce (when possible) the fundamental democratic principle of majority rule: a candidate who would defeat every other candidate in a one-on-one election is declared the *Condorcet winner*. Civitas eliminates the potential covert channel in a Condorcet ballot by encoding it into $\binom{C}{2}$ ballots, where C is the number of candidates. Each ballot is a three-way choice between (yes, no, indifferent) regarding two candidates i and j; for example, a "yes" choice means the voter prefers candidate i to candidate j. The voter is given a set of $\binom{C}{2}$ credentials, one for each possible $\langle i, j\rangle$ pair. The voter submits the $\binom{C}{2}$ credentials, along with the corresponding choices, with a cleartext label indicating the $\langle i, j\rangle$ pair. Any submission that is incomplete (i.e., missing an $\langle i, j\rangle$ pair) is eliminated during the first phase of tabulation. Subsequently, $\binom{C}{2}$ elections are effectively tabulated in parallel. When votes are finally decrypted, each of the $\binom{C}{2}$ choices for each voter can be summed to obtain a matrix of preferences that can be used to decide the winner. Because individual choices have been anonymized, the adversary is unable to learn more about a voter's low-order preferences than can be learned from the tally of the individual blocks.

8.2 Revoting

Voters may submit more than one vote per credential. The supervisor has the flexibility to specify a policy on how to tally such *revotes*. If revotes are not allowed, all votes submitted under duplicate credentials are eliminated during tabulation. If revotes are allowed, then the voter is responsible for including a zero-knowledge proof in later votes to indicate which earlier votes they replace. This proof requires knowledge of the secret credential and the choice used in both votes, so an adversary cannot revote on behalf of a voter using only information on the bulletin board.

8.3 Early returns

The Civitas blocking mechanism enables the production of *early returns*. A fraction of the blocks, randomly chosen, may be tabulated and used to predict the outcome of the entire election. Under appropriate assumptions, such as random assignment of voters to blocks by the registrar, this predictor may be correct with high probability.

9 Performance

A voting system is practical only if tabulation of the results can be completed in reasonable time, with reasonable cost and security. Civitas offers a tradeoff between these three factors, because tabulation can be completed more quickly by accepting higher cost or lower security. The goal of the performance study presented here is to show that practical tradeoffs exist between these factors, and that performance scales as predicted by the analysis in Section 6.

Notions of reasonable time, cost, and security may differ depending on the election or the observer. In current American elections, accurate predictions of election results are available within a few hours. Therefore, we choose a target tabulation time of five hours.

Currently, government elections in stable western democracies have a cost of \$1–\$3 per voter per election [39]. Changing from current methods to a voting system like Civitas would increase the cost of computing equipment, but could eliminate other costs, such as maintaining secure polling places and printing ballots. Section 9.3 considers the cost of running a Civitas election.

There are two important *security parameters*: the number of tabulation tellers (A) and the minimum number of voters within each block (K). As reasonable values for these parameters, we choose $K = 100$ and $A = 4$. Anonymity within 100 voters is comparable to what is available in current real-world elections, where results are tabulated at a precinct level and precinct workers may correlate voters with ballots. If voters are assigned randomly into blocks, as described in Section 6, then this level of anonymity may be even stronger than in real-world elections. Similarly, four (mutually distrusting) authorities offer better oversight than in many real-world elections.

9.1 Experiment design

We used Emulab [76] as an experiment testbed. The experiments ran on machines containing 3.0 GHz Xeon processors and 1 GB of RAM, networked on a 100 Mb LAN, and running Red Hat Linux 9.0 and Java 1.5.0_11. For RSA, AES, and SHA-512 implementations, we used the Bouncy Castle JCE provider 1.33. For the remaining cryptographic functionality, including El Gamal and various zero-knowledge proofs, we implemented our own cryptographic library, using GMP 4.2.1 for native modular exponentiation and multiplication implementations. The average time to perform a modular exponentiation was 4.7ms.

Each experiment simulated a complete election, including voter registration, voters casting votes to ballot boxes, and tabulation of the results. All the cryptographic protocols and operations described in Section 5 were carried out in full. Therefore, the behavior of the system should be representative of that of a real deployment.

All experiments used ballots with three choices ($C = 3$). Key lengths were constant: 1024-bit El Gamal keys, 2048-bit RSA keys, and 256-bit AES keys. No voters abstain, so $N \geq V$ and $M \geq K$.

Experiments were repeated three times, and the sample mean is reported in the graphs below. Error bars are not shown because the sample standard deviation is always less than 2.1% of the mean.

9.2 Time

The setup and voting phases of an election are not computationally intensive. Generation of keys and credentials in the setup phase scales linearly in the number of authorities and voters respectively, and can be conducted offline. During the voting phase, voters retrieve credential shares from registration tellers, and submit votes to ballot boxes. Experiments indicate that voting time is reasonable: about 350ms for a voter to acquire a credential

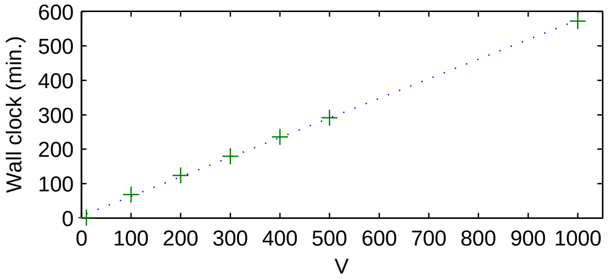

Figure 2: Tabulation time vs. voters ($K = 100, A = 4$)

share from a registration teller, and about 25ms to submit a vote to a ballot box. Thus, for four authorities, it takes a voter less than 1.5 seconds to retrieve credentials and submit a vote. From the registration teller's perspective, it takes approximately 230ms of CPU time to distribute a single voter's credential share. Thus, a registration teller could process more than 15,000 voters per hour.

The tabulation phase of an election, however, is computationally intensive. Therefore, the rest of this performance study focuses on tabulation.

Tabulation of each block is independent of other blocks, and combining the block tallies is a negligible part of the total tabulation time. This is demonstrated in Figure 2, where the tabulation tellers process blocks sequentially, and the number of voters V is a multiple of the block size $K = 100$. Civitas can tabulate ballots from 500 voters, on one set of machines, in the target tabulation time of five hours.

The independence of block tabulation can be exploited to decrease tabulation time in at least two ways. First, blocks can be tabulated in parallel, as discussed in Section 6. Given a set of tabulation teller machines for each block, the data in Figure 2 predict that tabulation could be completed in about 70 minutes, independent of V. Second, by computing early returns (as suggested in Section 8.3), a good predictor of election results can be obtained more quickly than the full tabulation. Because of the linear tradeoff between time and machines at the granularity of blocks, the remaining measurements in this study are for tabulation of a single block.

Security parameters A and K also affect tabulation time. Figure 3 shows that increasing the number of tellers increases tabulation time. Time increases nonlinearly because total communication increases quadratically in A. Changing anonymity parameter K affects tabulation time dramatically, because the plaintext equivalent tests take time proportional to K^2. Figure 4 shows this effect.

Chaff. In Civitas, voters can invent fake (i.e., invalid) credentials and submit ballots under those credentials to convince an adversary that they are voting as demanded. Voters can also submit extra, duplicate ballots under the same credential (see Section 8.2). We refer to invalid and duplicate ballots as *chaff* because they are eliminated during tabulation. Because

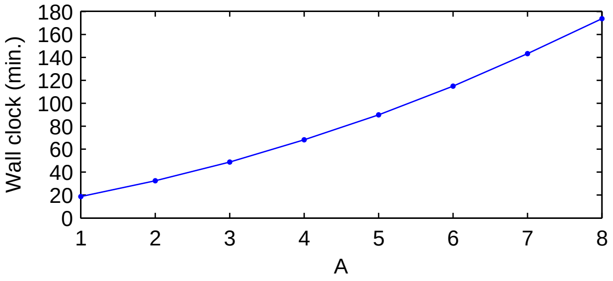

Figure 3: Tabulation time vs. authorities ($K = V = 100$)

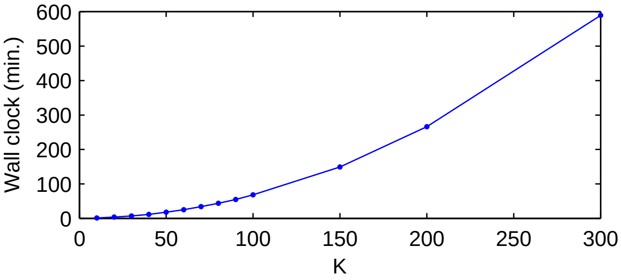

Figure 4: Tabulation time vs. anonymity ($V = K, A = 4$)

chaff increases the block size M, it increases the tabulation time, similarly to increasing anonymity parameter K. Figure 5 shows how tabulation time varies as a function of the percentage of chaff ballots in each block; with fraction c chaff (split between invalid and duplicate ballots) there are $M = \frac{V}{1-c}$ ballots in a block. All the other graphs in this study assume $c = .01$. The percentage of chaff ballots submitted by voters should be small, since coercion resistance discourages coercive attacks.

Memory usage. The memory footprint of Civitas is very small. With $M = 100$, the active set is no more than 7MB at any time during tabulation. The size of the active set scales linearly in M, so modern machines could easily fit tabulation in memory for substantially larger values of M and of K (since $K \leq M$).

9.3 Cost

A conservative way to estimate the cost of running an election would be to assume that the hardware is used for just one election and then discarded. Figure 2 shows that a secure election with 500 voters can be fully tabulated in about 5 hours. A dual-core version of our experiment machines is currently available for about $1,500, so the machine cost for this election is no more than $6,000, or $12 per voter. However, voting equipment is commonly

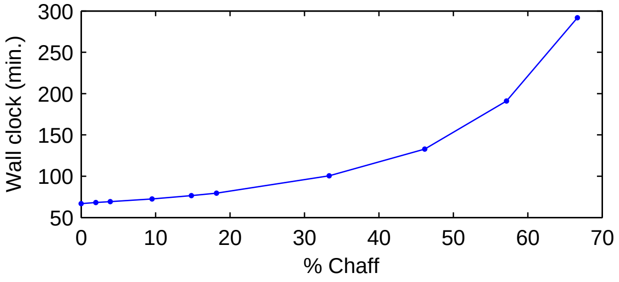

Figure 5: Tabulation time vs. chaff ($K = V = 100, A = 4$)

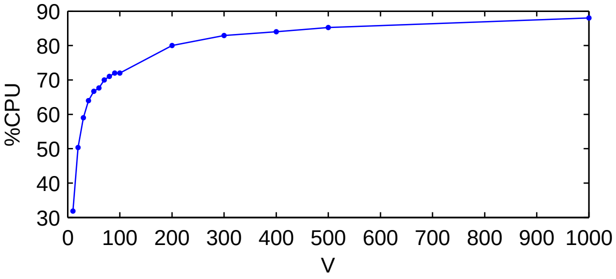

Figure 6: CPU utilization vs. voters ($K = 100, A = 4$)

reused for many elections, so this cost could be amortized.

If trust requirements permit leasing compute time from a provider, then costs can be reduced dramatically. One current provider offers computing time, at a rate of \$1 per CPU per hour, on processors similar in performance to our experiment machines [73]. At this rate, tabulation for 500 voters would cost about \$20, or \$0.04 per voter—clearly in the realm of practicality.

Another way to reduce the cost of tabulation would be to use multiple cores on a multicore CPU. Multicore architectures (and multiprocessor systems more generally) offer significant speedup for applications that are CPU-bound, have a small memory footprint, and can be split into parallel threads that interact frequently. Tabulation has all of these characteristics. The CPU utilization data of Figure 6 show that tabulation is CPU-bound.

Adjusting the security parameters of an election can also reduce cost. For example, Figure 4 demonstrates that halving K approximately quarters tabulation time. Thus, for a 10-hour tabulation time, with $K = 50$ and $A = 3$, the cost per voter should be about 10 times smaller than a 5-hour, $K = 100$, $A = 4$ election.

Based on this performance study, Civitas offers cost, time, and security that is practical for many real-world elections.

10 Discussion

All systems can be attacked. This section discusses possible attacks against Civitas, and corresponding defenses.

Attacks on the voter client. Much of the concern about the security of electronic voting has focused on vulnerabilities arising from direct-recording electronic (DRE) voting terminals [50, 48]. The analogous component of Civitas is the machine and software that implements the voter client. A key difference between the two kinds of systems is that in Civitas, the voter client is under the control of the voter. This has advantages and disadvantages with respect to security.

With DRE voting, the voter must use a voting machine that is provided by a local election authority, with little recourse against untrustworthy authorities. In contrast, with Civitas, voters are responsible for finding a voter client that they trust. Clearly, the voter client could be compromised at various levels. The machine, including the network connection, could potentially be controlled by the adversary. Further, any level of the software stack, including the Civitas software itself, could contain vulnerabilities that enable the adversary to compromise security.

It is reasonable to be concerned that voters' own machines are not secure enough to serve as a trustworthy voting client [64] and that voters will not scrutinize the voting software they use. However, with Civitas, voters are not constrained to use their own machine and software; they can use a machine provided by any organization that they trust, such as their political party, university, or other social organization. These organizations can also perform their own accreditation or even reimplementation of client hardware and software. This ability to transfer trust mitigates concerns about client security.

Furthermore, because the Civitas source code is publicly available, and is written in Jif, a type-safe programming language that controls information flows, it is easier to establish trust in the Civitas voter client software than in proprietary voting systems.

Attacks on registration. Voters may attempt to sell their identity so an adversary can register on their behalf. Or, an adversary may try to coerce a voter into revealing the voter's private key, which the adversary could then use to register as the voter.

Defending against these attacks seems to require authentication mechanisms that prevent the adversary from masquerading as the voter. For example, registration may need to be performed in person or by mail. This is a plausible solution, since registration may be done far in advance of the actual election. Moreover, if Trust Assumptions 2 and 5 are strengthened to honesty of tellers across multiple elections, credentials can be reused.

Attacks on the network. Civitas is secure in the presence of an adversary that has considerable control over the network. The adversary is assumed to be able to read and modify network traffic, except for the limitations expressed by Trust Assumption 2 (untappable channels) and Trust Assumption 3 (anonymous channels, justified in Section 4). Civitas secures all other channels with encryption, signing, and authentication protocols.

The untappable channel is used to prevent the following attack. Suppose the adversary can tap all channels to registration tellers and record the encrypted traffic between the voter and the registration tellers. Further suppose that the adversary can compromise the registration client so that it records all credential shares received from tellers. The adversary can then ask the client to reveal the plaintext credential shares corresponding to the encrypted network messages. The voter can no longer lie about any credential shares without being detected.

This attack assumes an adversary who has compromised the network and also can induce voters to register using a compromised client. So a possible defense is to employ better authentication mechanisms during registration, as discussed above.

The Civitas implementation of the registration client employs another defense: erasure of all credential shares once the voter's credential is constructed. This prevents the voter from reporting them to the adversary. The erasure of the share information is enforced within the code using erasure policies, as discussed in Section 7.

Attacks on election authorities. Trust Assumptions 2, 4, and 5 allow all but one of each of the types of election authorities to be compromised, while still guaranteeing coercion resistance and verifiability. Certain attacks may still be mounted by corrupting some small sets of authorities.

- A corrupt registration teller may fail to issue a valid credential share to a voter, and the voter necessarily cannot prove the (in)validity of a share to a third party. Defending against this could involve the voter and another election authority, perhaps an external auditor, jointly attempting to re-register the voter. The auditor could then attest to the misbehavior of a registration teller.

- The bulletin board may fail. Because bulletin board messages are signed, altered messages are detectable. Deletion of messages is an attack on availability, discussed below.

- A corrupt registrar may add bogus voters or remove legitimate voters from the electoral roll. Each tabulation teller can defend against this by refusing to tabulate unless the electoral roll is correct, according to some external policy.

- A corrupt supervisor could post an incorrect ballot design, stop an election early, or even attempt to simulate an election with only one real voter. All these can be defended against if the voters and tabulation tellers cease to participate in the election once the supervisor exhibits such behavior.

Attacks on availability. Civitas does not guarantee availability of either election authorities or the results of an election. However, the design of Civitas accommodates complementary techniques for achieving high availability.

To improve the availability of authorities (tellers, ballot boxes, and the bulletin board), they could be implemented as Byzantine fault-tolerant services [14, 63]. Also, the encryption scheme used by Civitas could be generalized from the current distributed scheme to a threshold scheme. This would enable election results to be computed even if some tabulation tellers permanently fail. However, an adversary would need to corrupt a smaller percentage of tellers to compromise the security of the election.

To defend against application-level denial-of-service attacks, standard techniques such as rate-limiting and puzzles could be employed. These will not, however, prevent the adversary from injecting a large number of invalid ballots, which would inflate tabulation time. A possible defense against this it to introduce a *block capability* that must be submitted with each vote. The registrar could issue each voter a unique, unforgeable capability for the voter's assigned block. The tabulation tellers could check this capability before performing the expensive plaintext equivalence tests. With this defense, the adversary must compromise a voter and learn the block capability to successfully inflate tabulation time for that block, and must repeat this attack for each block. If most blocks are not attacked and the margin of victory is sufficiently large, it would be possible (similar to the production of early returns in Section 8.3) to determine the result of the election by tabulating only the unattacked blocks. However, this mechanism is left for future work.

11 Related Work

Cryptographic voting schemes can be divided into three categories, depending on the technique used to anonymize votes: homomorphic encryption, blind signatures, and mix networks. Here we cite a few examples of each type.

In schemes based on *homomorphic encryption*, voters submit encrypted votes that are never decrypted [6, 22, 65, 38]. Rather, the set of submitted ciphertexts is combined (using some operation that commutes with the encryption function) to produce a single ciphertext containing the tally of the election, which is then decrypted. *Blind signature* schemes [32, 62, 61] split the election authority into an authenticator and tallier. The voter authenticates to the authenticator, presents a blinded vote, and obtains the authenticator's signature on the blinded vote. The voter unblinds the signed vote and submits it via an anonymized channel to the tallier. In *mix network* schemes, voters authenticate and submit encrypted votes [15, 66, 4, 54]. The votes are sent through a mix network to anonymize them, then the anonymized votes are decrypted.

Despite a plethora of voting schemes, few voting systems have actually been implemented, and none of these offers confidentiality guarantees as strong as in Civitas. Nor are any of these accompanied by security proofs. Sensus [23], based on a blind signature scheme known as FOO92 [32], offers no defense against coercion. Neither does EVOX [36], also based on FOO92. Both systems allow a single malicious election authority to vote on behalf of voters who abstain. EVOX-MA [28] addressed this by distributing the authority functionality. REVS [46, 52] extends EVOX-MA to tolerate failure of distributed components, but it does not address coercion. ElectMe [71] uses a scheme based on ideas from blind signature schemes, i.e., acquiring a signature of a part on a disguised vote to make the vote valid. Although ElectMe claims to provide coercion resistance, its coercion model is weaker than that of Civitas. In particular, its model does not allow the adversary to corrupt any election authorities. Further, if the adversary learns even the ciphertext of a voter's "ticket", the scheme fails to be receipt-free. ElectMe also fails to be universally verifiable: although voters can verify their ballots are recorded correctly, the computation of the tally is not publicly verifiable. Adder [49] implements a homomorphic scheme in which voters authenticate to a trusted "gatekeeper". If an adversary were to corrupt this gatekeeper, Adder would fail to be coercion-resistant. Four voting systems are currently listed as in development as part of the VoComp [70] competition, but it is not yet clear what security properties these systems will satisfy. Earlier work on one of these systems [51] claims to be coercion-free, but requires voters to sign votes, which appears susceptible to attacks in which coercers insist either that the voter abstain or submit a ballot prepared by the coercer. Kiayias [49] surveys several voting systems from the commercial world, including VoteHere [75]. These proprietary systems do not generally make their implementations publicly or freely available, nor do they appear to offer coercion resistance.

As an optimization of the JCJ scheme, Smith [72] proposes replacing plaintext equivalence tests with reencryption into a deterministic, distributed cryptosystem. Unfortunately, the construction proposed in that work is insecure. The proposed encryption function is $\mathsf{Enc}(m; z) = m^z$, where z is a secret key distributed among the tellers. However, this scheme is malleable: given $\mathsf{Enc}(m)$, an adversary can easily construct $\mathsf{Enc}(m^k)$ for any k. This leads to an attack on coercion resistance. To test whether s is a valid private credential, the adversary can inject a vote using s^2 as the private credential. After the proposed encryption function is applied during invalid credential elimination, the adversary can test whether any submitted credential is the square of any authorized credential; if so, then s is valid with high probability.

12 Conclusion

This paper describes a complete design, implementation, and evaluation of a secure remote voting system. To our knowledge, this has not been done before. Civitas satisfies stronger security properties than previously implemented voting systems and is secure against a stronger adversary, which is capable of corrupting most of the system. Although based on a previously known voting scheme, creating Civitas, and making its performance scale, led to the development of a secure registration protocol and a scalable ballot storage system. Civitas thus contributes to both the theory and practice of electronic voting systems.

The many cryptographic protocols required to implement Civitas (and other secure voting systems) are distributed throughout the literature, where they are described at various levels of abstraction. This paper (in Appendix B) makes a modest contribution by collecting these and presenting them in a concrete form that other system builders could use to reimplement Civitas.

Perhaps the most important contribution of this work is strong evidence that secure electronic voting is possible and even feasible, contrary to conventional wisdom. We are optimistic that voting systems constructed using principled techniques, like Civitas, will facilitate scientific and societal progress.

Acknowledgments

We thank Michael George, Nathaniel Nystrom, Tom Roeder, Peter Ryan, Fred B. Schneider, Hakim Weatherspoon, Lantian Zheng, and Lidong Zhou for discussions about this work. We also thank Jed Liu, Tudor Marian, and Tom Roeder for consultation on performance experiments.

Section 8.1 was previously presented at FEE'05 [20].

This work was supported by the Department of the Navy, Office of Naval Research, ONR Grant N00014-01-1-0968; Air Force Office of Scientific Research, Air Force Materiel Command, USAF, grant number F9550-06-0019; National Science Foundation grants 0208642, 0133302, 0430161, and CCF-0424422 (TRUST); and a grant from Intel Corporation. Michael Clarkson was supported by a National Science Foundation Graduate Research Fellowship; Andrew Myers was supported by an Alfred P. Sloan Research Fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of these organizations or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] Aslan Askarov, Daniel Hedin and Andrei Sabelfeld. Cryptographically-Masked Flows. In *Static Analysis Symposium*, pages 353–369, Seoul, Korea, August 2006.

[2] Association for Computing Machinery. SIG Elections. `www.acm.org/sigs/elections`, 2007.

[3] Jonathan Bannet, David W. Price, Algis Rudys, Justin Singer and Dan S. Wallach. Hack-a-Vote: Security Issues with Electronic Voting Systems. *IEEE Security & Privacy*, 2(1):32–37, January 2004.

[4] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Guillaume Poupard and Jacques Stern. Practical Multi-Candidate Election System. In *ACM Symposium on Principles of Distributed Computing*, pages 274–283, Newport, Rhode Island, August 2001.

[5] Mihir Bellare and Phillip Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, November 1993.

[6] Josh Daniel Cohen Benaloh. Verifiable Secret-Ballot Elections. Ph.D. Thesis, Yale University, September 1987.

[7] Dan Boneh, Antoine Joux and Phong Q. Nguyen. Why Textbook ElGamal and RSA Encryption Are Insecure. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 30–43, Kyoto, Japan, December 2000.

[8] Steven J. Brams and Peter C. Fishburn. Voting Procedures. In *Handbook of Social Choice and Welfare*, Volume 1, Chapter 4, Kenneth J. Arrow, Amartya K. Sen and Kotaro Suzumura, ed., Elsevier, 2002.

[9] Felix Brandt. Efficient Cryptographic Protocol Design Based on Distributed El Gamal Encryption. In *International Conference on Information Security and Cryptology*, pages 32–47, Seoul, Korea, December 2005.

[10] Brennan Center for Justice. The Machinery of Democracy: Voting System Security, Accessibility, Usability, and Cost. New York University, October 2006.

[11] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups. In *International Cryptology Conference (CRYPTO)*, pages 410–424, Santa Barbara, California, August 1997.

[12] Ran Canetti, Cynthia Dwork, Moni Naor and Rafail Ostrovsky. Deniable Encryption. In *International Cryptology Conference (CRYPTO)*, pages 90–104, Santa Barbara, California, August 1997.

[13] Ran Canetti and Rosario Gennaro. Incoercible Multiparty Computation. In *IEEE Symposium on Foundations of Computer Science*, pages 504–513, Burlington, Vermont, October 1996.

[14] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *ACM Symposium on Operating System Design and Implementation*, pages 173–186, New Orleans, Louisiana, February 1999.

[15] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[16] David Chaum and Torben Pryds Pedersen. Wallet Databases with Observers. In *International Cryptology Conference (CRYPTO)*, pages 89–105, Santa Barbara, California, August 1992.

[17] David Chaum, Peter Y. A. Ryan and Steve Schneider. A Practical Voter-Verifiable Election Scheme. In *European Symposium on Research in Computer Security*, pages 118–139, Milan, Italy, September 2005.

[18] Liming Chen and Algirdas Avizienis. N-Version Programming: A Fault Tolerance Approach to Reliability of Software Operation. In *International Symposium on Fault-Tolerant Computing*, 1978.

[19] Stephen Chong and Andrew C. Myers. Language-Based Information Erasure. In *IEEE Computer Security Foundations Workshop*, pages 241–254, Aix-en-Provence, France, June 2005.

[20] Michael R. Clarkson and Andrew C. Myers. Coercion-Resistant Remote Voting Using Decryption Mixes. In *Workshop on Frontiers in Electronic Elections*, Milan, Italy, September 2005.

[21] Ronald Cramer, Matthew Franklin, Berry Schoenmakers and Moti Yung. Multi-Authority Secret-Ballot Elections with Linear Work. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 72–83, Saragossa, Spain, May 1996.

[22] Ronald Cramer, Rosario Gennaro and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 103–118, Konstanz, Germany, May 1997.

[23] Laurie Faith Cranor and Ron K. Cytron. Sensus: A Security-Conscious Electronic Polling System for the Internet. In *IEEE Hawaii International Conference on Systems Science*, pages 561–570, Maui, Hawaii, January 1997.

[24] Debian Project. Debian Vote Engine (Devotee). `www.debian.org/vote`.

[25] Stephanie Delaune, Steve Kremer and Mark Ryan. Coercion-Resistance and Receipt-Freeness in Electronic Voting. In *IEEE Computer Security Foundations Workshop*, pages 28–42, Venice, Italy, July 2006.

[26] David L. Dill, Bruce Schneier and Barbara Simons. Voting and Technology: Who Gets to Count Your Vote?. *Communications of the ACM*, 46(8):29–31, August 2003.

[27] Roger Dingledine, Nick Mathewson and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, pages 303–320, San Diego, California, August 2004.

[28] Brandon William DuRette. Multiple Administrators for Electronic Voting. Bachelor's Thesis, Massachusetts Institute of Technology, May 1999.

[29] David Evans and Nathanael Paul. Election Security: Perception and Reality. *IEEE Security & Privacy*, 2(1):24–31, January 2004.

[30] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *International Cryptology Conference (CRYPTO)*, pages 186–194, Santa Barbara, California, August 1986.

[31] Stephanie Forrest, Anil Somayaji and David Ackley. Building Diverse Computer Systems. In *IEEE Workshop on Hot Topics in Operating Systems*, Cape Cod, Massachusetts, May 1997.

[32] Atsushi Fujioka, Tatsuaki Okamoto and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 244–251, Balatonfüred, Hungary, May 1992.

[33] Jun Furukawa. Efficient and Verifiable Shuffling and Shuffle-Decryption. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E88-A(1):172–188, 2005.

[34] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[35] Oded Goldreich, Silvio Micali and Avi Wigderson. How to Play Any Mental Game Or a Completeness Theorem for Protocols with Honest Majority. In *ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[36] Mark Herschberg. Secure Electronic Voting Over the World Wide Web. Master's Thesis, Massachusetts Institute of Technology, 1997.

[37] Boniface Hicks, Dave King, Patrick McDaniel and Michael Hicks. Trusted Declassification. In *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, pages 65–74, Ottawa, Canada, June 2006.

[38] Martin Hirt and Kazue Sako. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 539–556, Bruges, Belgium, May 2000.

[39] Internation Foundation for Election Systems. Getting to the CORE—A Global Survey on the Cost of Registration and Elections. United Nations Development Program, June 2006.

[40] International Organization for Standardization. Information Technology—Security Techniques—Key Management—Part 3: Mechanisms Using Asymmetric Techniques. ISO/IEC 11770-3, 1999.

[41] Markus Jakobsson and Ari Juels. Mix and Match: Secure Function Evaluation Via Ciphertexts. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 162–177, Kyoto, Japan, December 2000.

[42] Markus Jakobsson, Ari Juels and Ronald L. Rivest. Making Mix Nets Robust for Electronic Voting By Randomized Partial Checking. In *USENIX Security Symposium*, pages 339–353, San Francisco, California, August 2002.

[43] Markus Jakobsson, Kazue Sako and Russell Impagliazzo. Designated Verifier Proofs and Their Applications. In *International Conference on the Theory and Applications*

of Cryptographic Techniques (EUROCRYPT), pages 143–154, Saragossa, Spain, May 1996.

[44] David Jefferson, Aviel D. Rubin, Barbara Simons and David Wagner. A Security Analysis of the Secure Electronic Registration and Voting Experiment (SERVE). January 2004. `www.servesecurityreport.org/paper.pdf`.

[45] David Jefferson, Aviel D. Rubin, Barbara Simons and David Wagner. Analyzing Internet Voting Security. *Communications of the ACM*, 47(10):59–64, October 2004.

[46] Rui Joaquim, André Zúquete and Paulo Ferreira. REVS—A Robust Electronic Voting System. In *IADIS International Conference on e-Society*, Lisbon, Portugal, June 2003.

[47] Ari Juels, Dario Catalano and Markus Jakobsson. Coercion-Resistant Electronic Elections. In *Workshop on Privacy in the Electronic Society*, pages 61–70, Alexandria, Virginia, November 2005.

[48] Chris Karlof, Naveen Sastry and David Wagner. Cryptographic Voting Protocols: A Systems Perspective. In *USENIX Security Symposium*, 2005.

[49] Aggelos Kiayias, Michael Korman and David Walluck. An Internet Voting System Supporting User Privacy. In *Annual Computer Security Applications Conference*, pages 165–174, Miami Beach, Florida, December 2006.

[50] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin and Dan S. Wallach. Analysis of an Electronic Voting System. In *IEEE Symposium on Security and Privacy*, pages 27–42, Berkeley, California, May 2004.

[51] Mirosław Kutyłowski and Filip Zagórski. Coercion-Free Verifiable Internet Voting. In *Workshop on Electronic Voting and e-Government in the UK*, Edinburgh, Scotland, February 2006.

[52] Ricardo Lebre, Rui Joaquim, André Zúquete and Paulo Ferreira. Internet Voting: Improving Resistance to Malicious Servers in REVS. In *IADIS International Conference on Applied Computing*, Lisbon, Portugal, March 2004.

[53] Gavin Lowe. An Attack on the Needham-Schroeder Public Key Authentication Protocol. *Information Processing Letters*, 56(3):131–136, November 1995.

[54] Emmanouil Magkos, Mike Burmester and Vassilis Chrissikopoulos. Receipt-Freeness in Large-Scale Elections Without Untappable Channels. In *IFIP Conference on E-Commerce, E-Business, E-Government*, pages 683–694, Zürich, Switzerland, October 2001.

[55] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.

[56] Rebecca Mercuri. Statement on Electronic Voting. 2007. `www.notablesoftware.com/RMstatement.html`.

[57] Andrew C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *ACM Symposium on Principles of Programming Languages*, pages 228–241, San Antonio, Texas, January 1999.

[58] Andrew C. Myers and Barbara Liskov. Protecting Privacy Using the Decentralized Label Model. *ACM Transactions on Software Engineering and Methodology*, 9(4):410–442, October 2000.

[59] Andrew C. Myers, Lantian Zheng, Steve Zdancewic, Stephen Chong and Nathaniel Nystrom. Jif: Java Information Flow. `www.cs.cornell.edu/jif`, July 2001. Software Release.

[60] C. Andrew Neff. Verifiable Mixing (Shuffling) of ElGamal Pairs. April 2004. `www.votehere.org/vhti/documentation/egshuf-2.0.3638.pdf`.

[61] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka and Tatsuaki Okamoto. An Improvement on a Practical Secret Voting Scheme. In *Information Security Workshop*, pages 225–234, Kuala Lumpur, Malaysia, November 1999.

[62] Tatsuaki Okamoto. Receipt-Free Electronic Voting Schemes for Large Scale Elections. In *Security Protocols Workshop*, pages 25–35, Paris, France, April 1997.

[63] R. A. Peters. A Secure Bulletin Board. Master's Thesis, Technische Universiteit Eindhoven, June 2005.

[64] Aviel D. Rubin. Security Considerations for Remote Electronic Voting. *Communications of the ACM*, 45(12):39–44, December 2002.

[65] Kazue Sako and Joe Kilian. Secure Voting Using Partially Compatible Homomorphisms. In *International Cryptology Conference (CRYPTO)*, pages 411–424, Santa Barbara, California, August 1994.

[66] Kazue Sako and Joe Kilian. Receipt-Free Mix-Type Voting Scheme—A Practical Solution to the Implementation of a Voting Booth. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 393–403, Saint Malo, France, May 1995.

[67] Fred B. Schneider and Lidong Zhou. Distributed Trust: Supporting Fault-Tolerance and Attack-Tolerance. Cornell University, Technical Report 2004-1924, January 2004.

[68] Claus-Peter Schnorr. Efficient Signature Generation By Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[69] Claus-Peter Schnorr and Markus Jakobsson. Security of Signed ElGamal Encryption. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 73–89, Kyoto, Japan, December 2000.

[70] Alan T. Sherman, organizer. University Voting Systems Competition. `vocomp.org`, 2006. David Chaum, concept creator.

[71] Anna M. Shubina and Sean W. Smith. Design and Prototype of a Coercion-Resistant, Voter Verifiable Electronic Voting System. In *Conference on Privacy, Security and Trust*, pages 29–39, Fredericton, New Brunswick, Canada, October 2004.

[72] Warren D. Smith. New Cryptographic Election Protocol with Best-Known Theoretical Properties. In *Workshop on Frontiers in Electronic Elections*, Milan, Italy, September 2005.

[73] Sun Microsystems. Sun Grid Compute Utility: Users Guide. March 2007. `www.sun.com/service/sungrid/SunGridUG.pdf`.

[74] Dennis Volpano and Geoffrey Smith. A Type-Based Approach to Program Security. In *International Joint Conference on the Theory and Practice of Software Development*, pages 607–621, Lille, France, April 1997.

[75] VoteHere, Inc. Independent Audit Solutions for Validation and Verification of Election Results (Corporate Website). `votehere.com`, 1998. C. Andrew Neff, Chief Scientist.

[76] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *ACM Symposium on Operating System Design and Implementation*, pages 255–270, Boston, Massachusetts, December 2002.

A Security of Registration

The JCJ scheme assumed a registrar $\mathcal{R}$, which was trusted to generate and to maintain the confidentiality of private credentials. Civitas distributes this functionality and trust over a set REG of registration tellers. We must show that REG securely implements $\mathcal{R}$. In particular, the Civitas registration scheme is coercion-resistant if:

1. The private credentials generated by REG are uniformly distributed, and

2. Fake credentials are indistinguishable from valid credentials.

(These requirements were extracted from steps 2 and 5 of the JCJ simulation proof of coercion resistance.)

Assume that REG consists of only two tellers, RT_H and RT_C. Teller RT_H is an honest teller, whereas RT_C is corrupted by the adversary. This is without loss of generality, since Trust Assumption 2 assumes a single honest teller, and the actions of multiple corrupted tellers can be simulated by a single corrupt teller.

Let s_C be the private credential share generated for a voter by RT_C, and similarly for s_H. Then the voter's private credential is $s = s_H \cdot s_C$. Also, recall that shares are chosen from space $\mathcal{M} = QR_p$. The following lemma states that REG does generate uniformly distributed private credentials.

Lemma 1. $\forall s_C \,.\, \{s_H \leftarrow \mathcal{M} \;:\; s_H \cdot s_C\} = \{s \leftarrow \mathcal{M} \;:\; s\}$

Proof. Since $(\mathcal{M}, \cdot)$ is a group, there is a unique $s_H = s \cdot s_C^{-1}$. □

Toward proving the indistinguishability of fake and valid credentials, let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an indistinguishable public-key encryption scheme. Let $\mathsf{Enc}(m; K; r)$ denote the encryption of message m with public key K using random coins r. As before, let $\mathcal{M}$ be the message space of the scheme. Let $\mathcal{O}$ be the space of random coins, and let $\approx$ denote computational indistinguishability.

Let $(\mathsf{DVRP}, \widetilde{\mathsf{DVRP}})$ be a designated-verifier reencryption proof (DVRP) [38]. This is a proof that either the prover knows the private key k corresponding to the public key K of the verifier, or that two ciphertexts c and c' decrypt to the same plaintext. A valid DVRP, constructed as $\mathsf{DVRP}(K, c, c'; w)$, is produced by a prover using a witness w to prove c' is a reencryption of c. A *fake* DVRP, constructed as $\widetilde{\mathsf{DVRP}}(K, c, c'; k)$, is produced by a verifier, and proves knowledge of k. Intuitively, the key security property is that a valid proof is indistinguishable from a fake proof; formal definitions can be found in Jakobsson et al. [43].

To distribute a credential share s to voter V, an honest registration teller sends

$$(s, r', \mathsf{DVRP}(K_V, S, S'; w))$$

to the voter, where $S = \mathsf{Enc}(s; K_{\mathrm{TT}}; r)$ is the public share posted on the bulletin board, $S' = \mathsf{Enc}(s; K_{\mathrm{TT}}; r')$, and witness w is a function of r and r'. This establishes to the voter that S' is an encryption of s, and S' is a reencryption of S. The security of this registration scheme is fundamentally based on the ability, resulting from the DVRP, of voters to lie about the credential share. This is described by the following experiment:

```
$\mathit{RegExp}(n, b) =$
    $K_{\mathrm{TT}}, k_{\mathrm{TT}} \leftarrow \mathsf{Gen}(1^n);$
    $K_V, k_V \leftarrow \mathsf{Gen}(1^n);$
    $s \leftarrow \mathcal{M};\ r \leftarrow \mathcal{O};\ r' \leftarrow \mathcal{O};$
    $S \leftarrow \mathsf{Enc}(s; K_{\mathrm{TT}}; r);\ S' \leftarrow \mathsf{Enc}(s; K_{\mathrm{TT}}; r');$
    $P \leftarrow \mathsf{DVRP}(K_V, S, S'; w);$
    $\tilde{s} \leftarrow \mathcal{M};\ \tilde{r} \leftarrow \mathcal{O};$
    $\tilde{S} \leftarrow \mathsf{Enc}(\tilde{s};\ K_{\mathrm{TT}}; \tilde{r});$
    $\tilde{P} \leftarrow \widetilde{\mathsf{DVRP}}(K_V, S, \tilde{S}; k_V);$
    if $b$ then
        output $(K_{\mathrm{TT}}, K_V, S, s, r', P)$
    else
        output $(K_{\mathrm{TT}}, K_V, S, \tilde{s}, \tilde{r}, \tilde{P})$.
```

If $b = 1$, the voter tells the truth about his share, otherwise he lies and fakes a DVRP. The following lemma says that, to an adversary, these two cases are indistinguishable.

Lemma 2. $\{\mathit{RegExp}(n, 0)\}_{n \in \mathbb{N}} \approx \{\mathit{RegExp}(n, 1)\}_{n \in \mathbb{N}}$

Proof. (A sketch of a simple hybrid argument.) Define three hybrids:

$$\begin{array}{lclcl} H_0 & = & \{K_{\mathrm{TT}}, K_V, S, s, r', P\} & = & \mathit{RegExp}(n, 0) \\ H_1 & = & \{K_{\mathrm{TT}}, K_V, S, s, r', \tilde{P}'\} & & \\ H_2 & = & \{K_{\mathrm{TT}}, K_V, S, \tilde{s}, \tilde{r}, \tilde{P}\} & = & \mathit{RegExp}(n, 1), \end{array}$$

where $\tilde{P}' = \widetilde{\mathsf{DVRP}}(K_V, S, S'; k_V)$. By the definition of a designated-verifier proof, $H_0 \approx H_1$.

To show that $H_1 \approx H_2$, assume for contradiction that there exists a distinguisher D that has some non-negligible advantage in distinguishing H_1 and H_2. Using D, we can construct a machine A that breaks the indistinguishability of the encryption scheme, as follows. Machine A is given K_{TT}, challenges m_0 and m_1, and a ciphertext c that encrypts one of the challenges. It then constructs instances of H_1 and H_2, where $S = c$, $s = \tilde{s} = m_0$, and other values are sampled randomly, and asks D to distinguish them. Note that A can construct a fake DVRP because it generates the key pair (K_V, k_V).

By the polynomial transitivity of indistinguishability, we conclude $H_0 \approx H_2$. □

Let *Cred* be the information obtained by the adversary when the voter presents a valid credential and *FakeCred* be the information from a fake credential, where

$$\begin{array}{rcl} \mathit{Cred} & = & \{K_{\mathrm{TT}}, K_V, S_H, s_H, r'_H, P_H, S_C, s_C, r'_C, P_C\} \\ \mathit{FakeCred} & = & \{K_{\mathrm{TT}}, K_V, S_H, \tilde{s}_H, \tilde{r}_H, \tilde{P}_H, S_C, s_C, r'_C, P_C\}. \end{array}$$

Indistinguishability of valid and fake credentials, and the security of REG, then follows, as the voter can lie about share s_H.

Corollary 1. $\mathit{Cred} \approx \mathit{FakeCred}$

Proof. Immediate from Lemma 2. □

Theorem 1. *Registration is coercion-resistant.*

Proof. Immediate from Lemma 1 and Corollary 1. □

B Protocol Specification

In the following protocols, El Gamal cryptosystem parameters (p, q, g) are implicitly used as input to all protocols (except parameter generation itself). A hash function, denoted $hash$, is used in some protocols. In zero-knowledge proofs, $hash$ is assumed to implement a random oracle; elsewhere, it must be collision-resistant and one-way. Let $\mathcal{O}$ be a space of random bits of appropriate length in each context that it is used.

B.1 Encryption scheme

Civitas implements an El Gamal encryption scheme over message space QR_p, although it is convenient to consider this message space to be the isomorphic group $\mathbb{Z}_q$. Further, the zero element of the group is excluded because it has a degenerate encryption. Mapping from $\mathbb{Z}_q$ to QR_p is necessary because El Gamal encryption leaks the Legendre symbol of the plaintext [7].

The algorithms below describe standard, non-malleable, and distributed constructions of El Gamal encryption. An algorithm for distributed encryption is omitted because it is identical to the standard algorithm. The standard construction is due to El Gamal [34]; non-malleable, Schnorr and Jakobsson [69]; and distributed, Brandt [9].

ALGORITHM: El Gamal Parameter Generation

```
Due to:   Handbook of Applied Cryptography [55]
Input:    Security parameter $k$
Output:   Parameters $(p, q, g)$

  1. Select a random $k$-bit prime $q$; compute $p = 2q + 1$; repeat until $p$ is prime

  2. $h \leftarrow [2..p-2]$; repeat until $h^2 \not\equiv 1 \pmod{p}$ and $h^q \not\equiv 1 \pmod{p}$; $g = h^2 \bmod p$

  3. Output $(p, q, g)$
```

ALGORITHM: El Gamal Key Generation

```
Input:    El Gamal parameters $(p, q, g)$
Output:   Public key $y$, private key $x$

  1. $x \leftarrow \mathbb{Z}_q^*$

  2. $y = g^x \bmod p$

  3. Output $(y, x)$
```

ALGORITHM: EncodeQR

```
Input:   $m \in \mathbb{Z}_q^*$
Output:  $M \in QR_p$

  1. $M = \begin{cases} m & : \quad m^q \equiv 1 \pmod{p} \\ p - m & : \quad m^q \equiv -1 \pmod{p} \end{cases}$
```

ALGORITHM: DecodeQR

```
Input:   $M \in QR_p$
Output:  $m \in \mathbb{Z}_q^*$

  1. $m = \begin{cases} M & : \quad M < q \\ p - M & : \quad M > q \end{cases}$
```

ALGORITHM: El Gamal Encryption

```
Input:   Public key $y$, message $m \in \mathbb{Z}_q^*$
Output:  $\mathsf{Enc}(m; y)$

  1. $M = \mathsf{EncodeQR}(m)$
  2. $r \leftarrow \mathbb{Z}_q^*$
  3. Output $(g^r \bmod p, My^r \bmod p)$
```

ALGORITHM: El Gamal Reencryption

```
Input:   Public key $y$, ciphertext $c = (a, b)$
Output:  $\mathsf{Reenc}(c)$

  1. $r \leftarrow \mathbb{Z}_q^*$
  2. Output $(ag^r \bmod p, by^r \bmod p)$
```

ALGORITHM: El Gamal Decryption

Input: Private key x, ciphertext $c = (a, b)$
Output: $\mathsf{Dec}(c; x)$

```
1. $M = \frac{b}{a^x} \bmod p$
2. Output $\mathsf{DecodeQR}(M)$
```

ALGORITHM: Non-Malleable El Gamal Encryption

Input: Public key y, message $m \in \mathbb{Z}_q^*$
Output: $\mathsf{NMEnc}(m; y)$

```
1. $r, s \leftarrow \mathbb{Z}_q^*$
2. $(a, b) = \mathsf{Enc}(m; y; r)$
3. $c = hash(g^s \bmod p, a, b) \bmod q$
4. $d = (s + cr) \bmod q$
5. Output $(a, b, c, d)$
```

ALGORITHM: Non-Malleable El Gamal Decryption

Input: Private key x, ciphertext $e = (a, b, c, d)$
Output: $\mathsf{NMDec}(e; x)$

```
1. $V = hash(g^d a^{-c} \bmod p, a, b) \bmod q$
2. Abort if $V \neq c$
3. Output $\mathsf{Dec}((a, b); x)$
```

ALGORITHM: Commitment

Input: Message m
Output: $\mathsf{Commit}(m)$

```
1. Output $hash(m)$
```

PROTOCOL: Distributed El Gamal Key Generation

Public input: Parameters (p, q, g)
Output: Public key Y, public key shares y_i, private key shares x_i

1. S_i: $x_i \leftarrow \mathbb{Z}_q^*$; $y_i = g^{x_i} \bmod p$
2. S_i: Publish $\mathsf{Commit}(y_i)$
3. S_i: Barrier: wait until all commitments are available
4. S_i: Publish y_i and proof $\mathsf{KnowDlog}(g, y_i)$
5. S_i: Verify all commitments and proofs
6. $Y = \prod_i y_i \bmod p$ is the distributed public key
7. $X = \sum_i x_i \bmod q$ is the distributed private key

PROTOCOL: Distributed El Gamal Decryption

Public input: Ciphertext $c = (a, b)$, public key shares y_i
Private input (S_i): Private key share x_i
Output: $\mathsf{DistDec}(c; X)$

1. S_i: Publish decryption share $a_i = a^{x_i} \bmod p$ and proof $\mathsf{EqDlogs}(g, a, y_i, a_i)$
2. S_i: Verify all proofs
3. $A = \prod_i a_i \bmod p$
4. $M = \frac{b}{A} \bmod p$
5. Output $\mathsf{DecodeQR}(M)$

B.2 Zero-knowledge proofs

PROTOCOL: KnowDlog

Due to: Schnorr [68]
Principals: Prover P and Verifier V
Public input: h, v
Private input (P): x s.t. $v \equiv h^x \pmod{p}$

1. P: Compute:
   - $z \leftarrow \mathbb{Z}_q$
   - $a = h^z \bmod p$
   - $c = hash(v, a) \bmod q$
   - $r = (z + cx) \bmod q$
2. $P \rightarrow V$: a, c, r
3. V: Verify $h^r \equiv av^c \pmod{p}$.

PROTOCOL: EqDlogs

Due to: Chaum and Pedersen [16]
Principals: Prover P and Verifier V
Public input: f, h, v, w
Private input (P): x s.t. $v \equiv f^x \pmod{p}$ and $w \equiv h^x \pmod{p}$

1. P: Compute:
   - $z \leftarrow \mathbb{Z}_q$
   - $a = f^z \bmod p$
   - $b = h^z \bmod p$
   - $c = hash(v, w, a, b) \bmod q$
   - $r = (z + cx) \bmod q$
2. $P \rightarrow V$: a, b, c, r
3. V: Verify $f^r \equiv av^c \pmod{p}$ and $h^r \equiv bw^c \pmod{p}$.

PROTOCOL: DVRP

Due to:	Hirt and Sako [38]
Principals:	Prover P and Verifier V
Public input:	Public key h_V of V El Gamal ciphertexts $e = (x, y)$ and $e' = (x', y')$ Public key h under which e and e' are encrypted Let $E = (e, e')$
Private input (P):	ζ s.t. $x' \equiv xg^{\zeta} \pmod{p}$ and $y' \equiv yh^{\zeta} \pmod{p}$
Output:	$\mathsf{DVRP}(h_V, e, e'; \zeta)$

1. P: Compute:
   - $d, w, r \leftarrow \mathbb{Z}_q$
   - $a = g^d \bmod p$
   - $b = h^d \bmod p$
   - $s = g^w (h_V)^r \bmod p$
   - $c = hash(E, a, b, s) \bmod q$
   - $u = (d + \zeta(c + w)) \bmod q$
2. $P \rightarrow V$: c, w, r, u
3. V: Compute:
   - $a' = g^u / (x'/x)^{c+w} \bmod p$
   - $b' = h^u / (y'/y)^{c+w} \bmod p$
   - $s' = g^w (h_V)^r \bmod p$
   - $c' = hash(E, a', b', s') \bmod q$
4. V: Verify $c = c'$

PROTOCOL: FakeDVRP

Due to:	Hirt and Sako [38]
Principals:	Prover P
Public input:	Public key h_P of P El Gamal ciphertexts $e = (x, y)$ and $\tilde{e} = (\tilde{x}, \tilde{y})$ Public key h under which e and $\tilde{e}$ are encrypted Let $\tilde{E} = (e, \tilde{e})$
Private input (P):	Private key z_P of P
Output:	$\widetilde{\mathsf{DVRP}}(h_P, e, \tilde{e}; z_P)$

1. P: Compute:
   - $\alpha, \beta, \tilde{u} \leftarrow \mathbb{Z}_q$
   - $\tilde{a} = g^{\tilde{u}}/(\tilde{x}/x)^{\alpha} \bmod p$
   - $\tilde{b} = h^{\tilde{u}}/(\tilde{y}/y)^{\alpha} \bmod p$
   - $\tilde{s} = g^{\beta} \bmod p$
   - $\tilde{c} = \mathit{hash}(\tilde{E}, \tilde{a}, \tilde{b}, \tilde{s}) \bmod q$
   - $\tilde{w} = (\alpha - \tilde{c}) \bmod q$
   - $\tilde{r} = (\beta - \tilde{w})/(z_P) \bmod q$
2. $P \rightarrow V$: $\tilde{c}, \tilde{w}, \tilde{r}, \tilde{u}$
3. V: $\tilde{c}, \tilde{w}, \tilde{r}, \tilde{u}$ will verify as a DVRP, as above.

PROTOCOL: ReencPf

Due to: Hirt and Sako [38]
Principals: Prover P and Verifier V
Public input: $L \in \mathbb{N}$
$C = \{(u_i, v_i) \mid 1 \leq i \leq L\}$
$c = (u, v)$
Private input (P): $t \in [1..L]$ and $r \in \mathbb{Z}_q$ s.t. $(u, v) = (g^r u_t \bmod p, y^r v_t \bmod p)$

1. P: Compute:
   - $d_i, r_i \leftarrow \mathbb{Z}_q, i \in [1..L]$
   - $\{(a_i, b_i) \mid i \in [1..L]\}$, where:
     $a_i = (\frac{u_i}{u})^{d_i} g^{r_i} \bmod p$
     $b_i = (\frac{v_i}{v})^{d_i} y^{r_i} \bmod p$
   - $E = u, v, u_1, \ldots, u_L, v_1, \ldots, v_L$
   - $c = hash(E, a_1, \ldots, a_L, b_1, \ldots, b_L) \bmod q$
   - $w = (-rd_t + r_t) \bmod q$
   - $D = c - (\sum_{i \in [1..t-1, t+1..L]} d_i) \bmod q$
   - $R = (w + rd'_t) \bmod q$
   - $d_i^v = \begin{cases} d_i & : \quad i \neq t \\ D & : \quad i = t \end{cases}$
   - $r_i^v = \begin{cases} r_i & : \quad i \neq t \\ R & : \quad i = t \end{cases}$
2. $P \rightarrow V$: $(d_1^v, \ldots, d_L^v, r_1^v, \ldots, r_L^v)$
3. V: Compute:
   - $\{(a_i^v, b_i^v) \mid i \in [1..L]\}$, where:
     $a_i^v = (\frac{u_i}{u})^{d_i^v} g^{r_i^v} \bmod p$
     $b_i^v = (\frac{v_i}{v})^{d_i^v} y^{r_i^v} \bmod p$
   - $c' = hash(E, a_1^v, \ldots, a_L^v, b_1^v, \ldots, b_L^v) \bmod q$
   - $D' = \sum_{i \in [1..L]} d_i^v \bmod q$
4. V: Verify $c' = D'$

PROTOCOL: VotePf

Due to:	Camenisch and Stadler [11]
Principals:	Prover P and Verifier V
Public input:	Encrypted credential (a_1, b_1)
	Encrypted choice (a_2, b_2)
	Vote context ctx (election identifier, etc.)
	Let $E = (g, a_1, b_1, a_2, b_2, ctx)$
Private input (P):	α_1, α_2 s.t. $a_i \equiv g^{\alpha_i} \pmod{p}$

1. P: Compute:
   - $r_1, r_2 \leftarrow \mathbb{Z}_q$
   - $c = hash(E, g^{r_1} \bmod p, g^{r_2} \bmod p) \bmod q$
   - $s_1 = (r_1 - c\alpha_1) \bmod q$
   - $s_2 = (r_2 - c\alpha_2) \bmod q$
2. $P \rightarrow V$: c, s_1, s_2
3. V: Compute $c' = hash(E, g^{s_1} a_1^c, g^{s_2} a_2^c) \bmod q$
4. V: Verify $c = c'$

B.3 Main protocols

PROTOCOL: Plaintext Equivalence Test (PET)

Due to:	Jakobsson and Juels [41]
Principals:	Tabulation tellers TT_i
Public input:	$c_j = \mathsf{Enc}(m_j; K_{\text{TT}}) = (a_j, b_j)$ for $j \in \{1, 2\}$
Private input (TT_i):	Private key share x_i
	Let $R = (d, e) = (a_1/a_2, b_1/b_2)$
Output:	If $m_1 = m_2$ then 1 else 0

1. TT_i: $z_i \leftarrow \mathbb{Z}_q^*$; $(d_i, e_i) = (d^{z_i}, e^{z_i})$
2. TT_i: Publish $\mathsf{Commit}(d_i, e_i)$
3. TT_i: Barrier: wait until all commitments are available
4. TT_i: Publish (d_i, e_i) and proof $\mathsf{EqDlogs}(d, e, d_i, e_i)$
5. TT_i: Verify all commitments and proofs
6. Let $c' = (\prod_i d_i, \prod_i e_i)$
7. All TT: Compute $m' = \mathsf{DistDec}(c')$ using private key shares
8. If $m' = 1$ then output 1 else output 0

ALGORITHM: Mix

Due to:	Jakobsson, Juels, and Rivest [42]
Input:	List L of ciphertexts $[c_1, \ldots, c_m]$, verification direction $\mathit{dir} \in \{\mathit{in}, \mathit{out}\}$
Output:	RPC reencryption mix of L

1. $\pi \leftarrow$ Space of permutations over m elements
2. If $\mathit{dir} = \mathit{in}$ then $p = \pi$ else $p = \pi^{-1}$
3. $r_1 \leftarrow \mathbb{Z}_q^* \ \ldots;\ r_m \leftarrow \mathbb{Z}_q^*;\ w_1 \leftarrow \mathcal{O};\ \ldots;\ w_m \leftarrow \mathcal{O}$
4. $L_R = [\mathsf{Reenc}(c_{\pi(1)}; r_1), \ldots, \mathsf{Reenc}(c_{\pi(m)}; r_m)]$
5. $L_C = [\mathsf{Commit}(w_1, p(1)), \ldots, \mathsf{Commit}(w_m, p(m))]$
6. Output L_R, L_C

PROTOCOL: MixNet

Due to:	Jakobsson, Juels, and Rivest [42]
Principals:	Tabulation tellers $\mathrm{TT}_1, \ldots, \mathrm{TT}_n$
Public input:	List L of ciphertexts $[c_1, \ldots, c_m]$
Output:	Anonymization of L

1. For i = 1 to n, sequentially:
   (a) Let $\mathsf{Mix}_{i,j}$ denote the j^{th} mix performed by the i^{th} teller.
   Define $\mathsf{Mix}_{0,2}.L_R = L$.
   (b) Let $L_1 = \mathsf{Mix}_{i-1,2}.L_R$
   (c) TT_i : Publish $\mathsf{Mix}_{i,1} = \mathsf{Mix}(L_1, out)$
   (d) Let $L_2 = \mathsf{Mix}_{i,1}.L_R$
   (e) TT_i : Publish $\mathsf{Mix}_{i,2} = \mathsf{Mix}(L_2, in)$
   (f) TT_i : $q_i \leftarrow \mathcal{O}$; publish $\mathsf{Commit}(q_i)$
2. All TT_i: Publish q_i; verify all other tellers' commitments
3. Let $Q = hash(q_1, \ldots, q_n)$
4. All TT_i:
   (a) Let $Q_i = hash(Q, i)$
   (b) For j in $[1..m]$, publish r_j, w_j, and $p(j)$ from $\mathsf{Mix}_{i,1+Q_i[j]}$
   (c) Verify all commitments (w and p) and reencryptions (r) from all other tellers, i.e.:
      i. Verify w_j and $p(j)$ against $\mathsf{Mix}_{i,1+Q_i[j]}.L_C[j]$
      ii. If $Q_i[j] = 0$ then verify $\mathsf{Reenc}(\mathsf{Mix}_{i-1,2}.L_R[p(j)]; r_j) = \mathsf{Mix}_{i,1}.L_R[j]$,
      else if $Q_i[j] = 1$ then verify $\mathsf{Reenc}(\mathsf{Mix}_{i,1}.L_R[j]; r_j) = \mathsf{Mix}_{i,2}.L_R[p(j)]$
5. Output $\mathsf{Mix}_{n,2}.L_R$

PROTOCOL: Register

Due to:	Needham-Schroeder-Lowe [53] (steps 1–8) Our own adaptation of ideas from [22, 38, 47] (steps 9–11)
Principals:	Registration teller RT_i and Voter V
Public input:	$K_{\mathrm{TT}}, K_{\mathrm{RT}_i}, K_V$, election identifier eid, voter identifier vid, teller identifier rid, public credential $S_i = \mathsf{Enc}_{\mathrm{EG}}(s_i; K_{\mathrm{TT}}; r)$
Private input (RT_i):	Private credential s_i and encryption factor $r \in \mathbb{Z}_q^*$
Private input (V):	k_V

1. V: $N_V \leftarrow \mathcal{N}$
2. $V \rightarrow \mathrm{RT}_i$: $\mathsf{Enc}_{\mathrm{RSA}}(eid, vid, N_V; K_{\mathrm{RT}_i})$
3. RT_i: Verify that vid is a voter in eid and lookup credential s_i
4. RT_i: $N_R \leftarrow \mathcal{N}$; $k \leftarrow \mathsf{Gen}_{\mathrm{AES}}(1^l)$
5. $\mathrm{RT}_i \rightarrow V$: $\mathsf{Enc}_{\mathrm{EG}}(rid, N_R, N_V, k; K_V)$
6. V: Verify rid and N_V
7. $V \rightarrow \mathrm{RT}_i$: N_R
8. RT_i: Verify N_R
9. RT_i: $r' \leftarrow \mathbb{Z}_q^*$; $w = r' - r$; $S_i' = \mathsf{Enc}_{\mathrm{EG}}(s_i; K_{\mathrm{TT}}; r')$
10. $\mathrm{RT}_i \rightarrow V$: $\mathsf{Enc}_{\mathrm{AES}}(s_i, r', \mathsf{DVRP}(K_V, S_i, S_i'; w)); k)$
11. V: Verify DVRP against S_i from bulletin board

To register, voter V completes protocol Register with each registration teller RT_i. After constructing a complete private credential $s = \prod_i s_i$, the voter may erase all shares s_i, DVRPs, and session key k.

In protocol Register, $\mathcal{N}$ is the space of nonces and l is the security parameter for AES. As discussed in Section 5, the use of RSA could be replaced by another cryptosystem, but the voter's El Gamal key is necessary for the DVRP. The use of AES could similarly be replaced by another cryptosystem, perhaps El Gamal in a block mode, or even a construction of deniable encryption [12].

The authentication protocol used in steps 1–8 of Register is not strictly an implementation of Needham-Schroeder-Lowe because step 7 is not encrypted under K_{RT_i}. This is admissible because nonce N_R is not used to construct the session key k. In this respect, the authentication protocol is similar to ISO/IEC 11770-3 Key Transport Mechanism 6 [40].

PROTOCOL: Vote

Due to:	Juels, Catalano, and Jakobsson [47]
Principals:	Voter V, ballot boxes
Public input:	K_{TT}, vote context ctx, well-known candidate ciphertext list $C = (c_1, \ldots, c_L)$
Private input (V):	Private credential s and candidate choice c_t for some t

1. V: $r_s \leftarrow \mathbb{Z}_q^*$; $es = \mathsf{Enc}(s; K_{\mathrm{TT}}; r_s)$
2. V: $r_v \leftarrow \mathbb{Z}_q^*$; $ev = \mathsf{Reenc}(c_t; K_{\mathrm{TT}}; r_v)$
3. V: $vote = (es, ev, \mathsf{VotePf}(es, et, ctx, r_s, r_v), \mathsf{ReencPf}(C, ev, t, r_v))$
4. $V \rightarrow$ ballot boxes: $vote$

PROTOCOL: Tabulate

Due to:	Juels, Catalano, and Jakobsson [47]
Principals:	Tabulation tellers $\mathrm{TT}_1, \ldots, \mathrm{TT}_n$, Bulletin board ABB Ballot boxes $VBB_1, \ldots,$ VBB_m, Supervisor Sup
Public input:	K_{TT}, contents of ABB
Private input (TT_i):	Private key share x_i of K_{TT}
Output:	Election tally

1. All VBB_i: Post Commit(received votes) on ABB

2. Sup: Post endorsement of all received VBB commitments

3. All TT_i: Proceed sequentially through the following phases. Each phase has a list (e.g., A, B, etc.) as output. In each phase that uses such a list as input, verify that all other tellers are using the same list. Use ABB as a public broadcast channel for any subprotocol that requires publication of a value; all posts to ABB must (as usual) be signed, and all messages retrieved from it should have their signatures verified.

   Retrieve Votes. Retrieve all votes from all endorsed VBBs. Verify the VBB commitments. Let the list of votes be A.

   Check Proofs. Verify all $\mathsf{VotePfs}$ and $\mathsf{ReencPfs}$ in retrieved votes. Eliminate any votes with an invalid proof. Let the resulting list be B.

   Duplicate Elimination. Run $\mathsf{PET}(s_i, s_j)$ for all $1 \leq i < j \leq |B|$, where s_x is the encrypted credential in vote $B[x]$. Eliminate any votes for which the PET returns 1 according to a revoting policy; let the remaining votes be C.

   Mix Votes. Run $\mathsf{MixNet}(C)$ and let the anonymized vote list be D.

   Mix Credentials. Retrieve all credentials from ABB and let this list be E. Run $\mathsf{MixNet}(E)$ and let the anonymized credential list be F.

   Invalid Elimination. Run $\mathsf{PET}(s_i, t_j)$ for all $1 \leq i \leq |F|$ and $1 \leq j \leq |D|$, where $s_i = F[i]$ and $t_j = D[j]$. Eliminate any votes (from D) for which the PET returns 0. Let the remaining votes be G.

   Decrypt. Run $\mathsf{DistDec}$ on all encrypted choices in G. Output the decryptions as H, the votes to be tallied.

   Tally. Compute tally of H using an election method specified on ABB by Sup. Verify tally from all other tellers.

4. Sup: Endorse tally