Civitas: Toward a Secure Voting System

Michael R. Clarkson   Stephen Chong   Andrew C. Myers

Department of Computer Science
Cornell University
{clarkson,schong,andru}@cs.cornell.edu

Computing and Information Science Technical Report
http://hdl.handle.net/1813/7875
(previously TR 2007-2081)

May 2007
Revised August 2007, November 2007, May 2008

Civitas: Toward a Secure Voting System

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

Abstract

Civitas is the first electronic voting system that is coercion-resistant, universally and voter verifiable, and suitable for remote voting. This paper describes the design and implementation of Civitas. Assurance is established in the design through security proofs, and in the implementation through information-flow security analysis. Experimental results give a quantitative evaluation of the tradeoffs between time, cost, and security.

1 Introduction

Electronic voting is now a reality—and so are the many errors and vulnerabilities in commercial electronic voting systems [4, 12, 61, 91]. Voting systems are hard to make trustworthy because they have strong, conflicting security requirements:

- *Integrity* of election results must be assured so that all voters are convinced that votes are counted correctly. Any attempt to corrupt the integrity of an election must be detected and correctly attributed.
- *Confidentiality* of votes must be assured to protect voters' privacy, to prevent selling of votes, and to defend voters from coercion.

Integrity is easy to obtain through a public show of hands, but this destroys confidentiality. Confidentiality can be obtained by secret ballots, but this fails to assure integrity. Because of the civic importance of elections, violations of these requirements can have dramatic consequences.

Many security experts have been skeptical about electronic voting [32, 36, 54, 66, 78], arguing that assurance in electronic voting systems is too hard to obtain and that their deployment creates unacceptable risks. Our work, however, was inspired by the possibility that electronic voting systems could be more trustworthy than their non-electronic predecessors. This paper describes and evaluates Civitas, the prototype system we built to explore that possibility. Although not yet suitable for deployment in national elections, Civitas enforces *verifiability* (an integrity property) and

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 and an Intel PhD 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.

coercion resistance [58] (a confidentiality property). Civitas does not rely on trusted supervision of polling places, making it a *remote* voting system.

To obtain assurance in the security of Civitas, we employed principled techniques:

- **Security proofs.** The design of Civitas refines a cryptographic voting scheme[1] due to Juels, Catalano, and Jakobsson [58], who proved their scheme secure; we extend the proof to account for our changes.

- **Secure information flow.** The implementation of Civitas is in Jif [67, 69], a language which enforces information-flow security policies.

This validation of the design and implementation supports our argument that Civitas is secure.

The security provided by Civitas is not free. Tradeoffs exist between the level of security provided by Civitas tabulation, the time required for tabulation, and the monetary cost of tabulation. To better understand these tradeoffs, we studied the performance of Civitas. The results reveal that (with reasonable security and time parameters), the marginal cost of tabulation is as low as 4¢ per voter. Since the current cost of a government election in a stable Western democracy is $1 to $3 per voter [49], Civitas can provide increased security at little additional cost.

Developing Civitas led to several contributions:

- A provably secure voter registration protocol, which distributes trust over a set of registration authorities.

- A scalable design for vote storage that ensures integrity without expensive fault tolerance mechanisms.

- A performance study demonstrating the scalability of secure tabulation.

- A coercion-resistant construction for implementing a ranked voting method.

- A concrete, publicly available specification of the cryptographic protocols required to implement a coercion-resistant, verifiable, remote voting scheme. This specification leverages many results in the cryptographic and voting literature.

Moreover, Civitas is the first voting system to implement a scheme proved to satisfy coercion resistance and verifiability. Thus, Civitas takes an important step toward bringing secure electronic voting to reality.

We proceed as follows. Section 2 discusses the Civitas security model. The design of Civitas is presented in Section 3. Section 4 evaluates the security of Civitas. The implementation of cryptographic components is described in Section 5, and the scalability of tabulation is analyzed in Section 6. The Jif implementation is described in Section 7. Ranked voting methods are discussed in Section 8. Section 9 presents our performance study. Related work is reviewed in Section 10, and some remaining challenges are identified in Section 11. Section 12 concludes.

[1] For clarity, we define voting *systems* as implementations, voting *schemes* as cryptographic protocols, and voting *methods* as algorithms that aggregate voters' preferences to produce a collective decision.

2 Security Model

The Civitas security model comprises the environment in which Civitas is used, the security properties we require Civitas to satisfy, and the capabilities we ascribe to the *adversary* attempting to subvert those properties.

Remote voting. Electronic voting systems are often designed for *supervised* voting, which assumes trusted human supervision of the voters, procedures, hardware, and software in polling places. But this contradicts society's trend toward enabling interactions from anywhere at any time. For example, voters in the state of Oregon now vote only by postal mail, and all states receive a substantial fraction—enough to change the outcome of many elections—of their ballots by mail as absentee ballots. As another example, Internet voting is increasingly used by groups such as Debian [28], the ACM [3], and the IEEE [48]. Estonia even conducts legally binding national elections using the Internet [35].

Postal voting and Internet voting are instances of remote voting, which does not assume trusted supervision of polling places. Remote voting is thus a more general problem, and a harder problem, than supervised voting. Because of the evident interest in remote voting, we believe that remote voting is the right problem to solve. One of our goals was therefore to strike a reasonable compromise between enabling remote voting and guaranteeing strong security properties. This compromise led to two requirements. First, in some circumstances, voters must register at least partly in person. Second, voters must trust the computational device they use to submit votes—though unlike conventional supervised voting, in which voters must trust the particular device supplied by their local election authorities, Civitas enables each voter to choose a supplier and device. We discuss these requirements in Section 4.

Security properties. To fulfill the integrity requirement of Section 1, we require Civitas to satisfy:

> **Verifiability.** *The final tally is verifiably correct. Each voter can check that their own vote is included in the tally* (voter verifiability). *Anyone can check that all votes cast are counted, that only authorized votes are counted, and that no votes are changed during counting* (universal verifiability).[2]

We define "verifiability" informally for simplicity, but Civitas satisfies the formal definition given by Juels et al. [58].[3]

Verifiability improves upon the integrity properties commonly offered by real-world voting systems. For example, real-world systems rarely allow individual voters to verify that their own votes were included in the tally, or to verify the tally themselves. As another example, the commercial electronic voting systems currently deployed in California offer no guarantees that votes are counted correctly [91].

To fulfill the confidentiality requirement of Section 1, a voting system might guarantee *anonymity*, meaning that the information released by the system never reveals how a voter voted. However, for remote voting, anonymity is too weak. Voters might gain additional information during voting that could enable the buying and selling of votes. Such information could also be used to coerce voters.

[2] Universal verifiability was originally defined by Sako and Kilian [81].

[3] Verifiability could be formulated as the correctness property of secure multi-party computation [42]. Intuitively, this requires that no adversary can change the results of tabulation to be different than if all votes were announced and tabulated publicly.

In remote voting, the coercer could even be the voter's employer or domestic partner, physically present with the voter and controlling the entire voting process. Against such coercers, it is necessary to ensure that voters can appear to comply with any behavior demanded of them. Further, confidentiality must be maintained even when voters collude with the adversary.

Thus, for confidentiality, we require Civitas to satisfy:

> **Coercion Resistance.** *Voters cannot prove whether or how they voted, even if they can interact with the adversary while voting.*[4]

We define "coercion resistance" informally[5] for simplicity, but Civitas again satisfies the formal definition given by Juels et al. [58].[6] This formal definition requires Civitas to defend against attacks in which the adversary demands secrets known to the voter, and attacks in which the adversary demands that the voter submits a value chosen by the adversary. This value might be a legitimate vote or a random value. The adversary may even demand that the voter abstain by submitting no value at all.[7]

A third security requirement that could be added is *availability* of the voting system and tabulation results. Although this would be essential for a national voting system, we do not require our prototype to satisfy any availability property. Some aspects of availability, such as fault tolerance, could be addressed by well-known techniques. Other aspects, such as defending against selective denial-of-service attacks intended to disenfranchise particular groups of voters, are open problems.

Threat model. We require Civitas to be secure with respect to an adversary (essentially due to Juels et al. [58]) with the following capabilities:

- The adversary may corrupt a threshold (made precise in Section 4) of the *election authorities*, mutually distrusting agents who conduct an election. Agents might be humans, organizations, or software components.
- The adversary may coerce voters, demand their secrets, and demand any behavior of them—remotely or in the physical presence of voters. But the adversary may not control a voter throughout an entire election, otherwise the voter could never register or vote.
- The adversary may control all public channels on the network. However, we also 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.[8]
- The adversary may perform any polynomial-time computation.

[4] Removing interaction with the adversary results in *receipt-freeness*, a weaker property originally defined by Benaloh [8].

[5] "Coercion resistance" is used informally throughout the literature. Juels et al. [58] and Delaune et al. [29] give formal definitions in the computational and symbolic models, respectively, of cryptography. The informal definition given above is consistent with both.

[6] Coercion resistance could be formulated as the privacy property of secure multi-party computation. Intuitively, this requires that no adversary can learn any more about votes than is revealed by the results of tabulation.

[7] Note that the requirement to defend voters from forced-abstinence attacks is incompatible with a public record of who has voted.

[8] An untappable channel must provide perfect secrecy, perhaps by being physically untappable or by implementing a one-time pad.

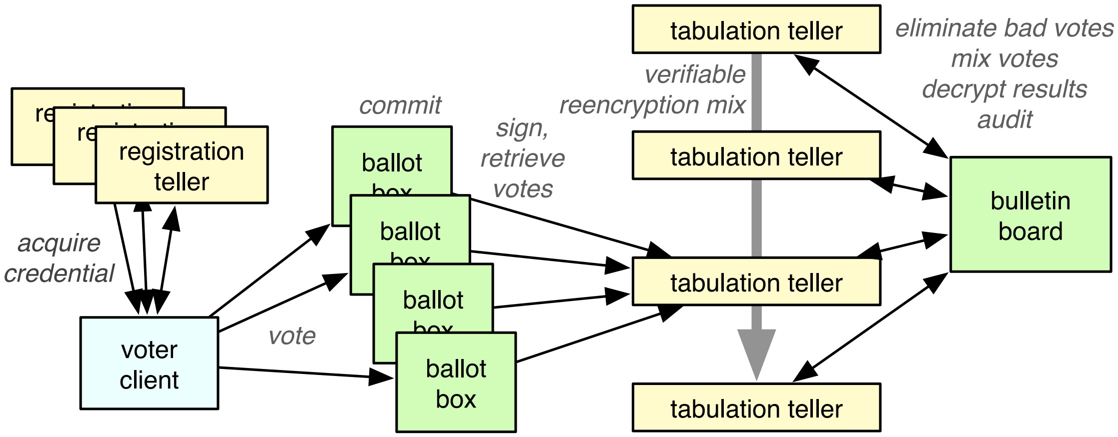

Figure 1: Civitas architecture

3 Design

Civitas refines and implements a voting scheme, which we refer to as *JCJ*, developed by Juels, Catalano, and Jakobsson [58]. The differences between our design and JCJ are discussed in Section 10.

3.1 Agents

There are five kinds of agents in the Civitas voting scheme: a supervisor, a registrar, voters, registration tellers, and tabulation tellers. Some of these are depicted in Figure 1. The agents other than voters are election authorities:

- The *supervisor* administers an election. This includes specifying the ballot design and the tellers, and starting and stopping the election.
- The *registrar* authorizes voters.
- *Registration tellers* generate the credentials that voters use to cast their votes.
- *Tabulation tellers* tally votes.

These agents use an underlying *log* service that implements publicly readable, insert-only storage. Integrity of messages in a log is ensured by digital signatures. Agents may sign messages they insert, ensuring that the log service cannot forge new messages. The log service must sign its responses to reads, ensuring that attempts to present different views of log contents to different readers can be detected. Multiple instances of the log service are used in a single election. One instance, called the *bulletin board*, is used by election authorities to record all the information needed for verifiability of the election. The remaining instances, called *ballot boxes*, are used by voters to cast their votes.[9]

[9] In our prototype, the log service instances are centralized systems provided by the election authorities—the bulletin board by the supervisor, and one ballot box by each tabulation teller. But instances could be made distributed systems to improve availability, and instances could be provided by agents other than the election authorities.

3.2 Setup phase

First, the supervisor creates the election by posting the ballot design on an empty bulletin board. The supervisor also identifies the tellers by posting their individual public keys.[10]

Second, the registrar posts the *electoral roll*, containing identifiers (perhaps names or registration numbers) for all authorized voters, along with the voters' public keys. Each voter is assumed to have two keys, a *registration* key and a *designation* key, whose uses are described below.

Third, the tabulation tellers collectively generate a public key for a distributed encryption scheme and post it on the bulletin board. Decryption of messages encrypted under this 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 an asymmetric cryptosystem, credentials are pairs of a public value and a private value. All public credentials are posted on the bulletin board, and each registration teller stores a share of each private credential. Private credentials can be forged or leaked only if all registration tellers collude.

3.3 Voting phase

Voters register to acquire their private credentials. Each registration teller authenticates a voter using the voter's registration key. The teller and voter then run a protocol, using the voter's designation key, that releases the teller's share of the voter's private credential to the voter. The voter combines all of these shares to construct a private credential.

Voting may take place immediately, or a long time after registration. 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 submission does not require either of the voter's keys.) Replication of the vote across the ballot boxes is used to guarantee availability of the vote for tabulation.

Resisting coercion. The key idea (due to Juels et al. [58]) that enables voters to resist coercion, and defeats vote selling, is that voters can substitute *fake* credentials for their real credentials, then behave however the adversary demands. For example:

If the adversary demands that the voter...	**Then the voter...**
Submits a particular vote	Does so with a fake credential.
Sells or surrenders a credential	Supplies a fake credential.
Abstains	Supplies a fake credential to the adversary and votes with a real one.

To construct a fake credential, the voter locally runs an algorithm to produce fake private credential shares that, to an adversary, are indistinguishable from real shares. The faking algorithm requires the voter's private designation key. The voter combines these shares to produce a fake private credential; the voter's public credential remains unchanged.

[10] A real-world deployment of Civitas would need a public-key infrastructure to certify keys.

Revoting. Voters might 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, then all votes submitted under duplicate credentials are eliminated. If revotes are allowed, then the voter must include a proof in later votes to indicate which earlier votes are being replaced. This proof must demonstrate knowledge of the credential and choice used in both votes, preventing an adversary from revoting on behalf of a voter.

Ballot design. Civitas is compatible with the use of any ballot design for which a proof of well-formedness is possible. Our prototype supports the use of ballots in which voters may choose a single candidate (plurality voting), any subset of candidates (approval voting), or a ranking of the candidates (ranked voting). However, ranked voting introduces covert channels that enable attacks on coercion resistance. We discuss this vulnerability, and how to eliminate it, in the accompanying technical report [23].[11]

Write-in votes could also be supported by Civitas, since any write-in could be considered well-formed. However, write-ins also enable attacks on coercion resistance.[12] To our knowledge, it is not possible to eliminate this vulnerability, so we chose not to implement write-ins in our prototype.

3.4 Tabulation phase

The tabulation tellers collectively tally the election:

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

2. **Verify proofs.** The tellers check each vote to verify the proof of well-formedness. Any vote with an invalid proof is discarded. (For efficiency, our implementation actually merges this with the next step.)

3. **Eliminate duplicates.** At most one vote is retained for each credential. Votes with duplicate credentials are eliminated according to the revoting policy.

4. **Anonymize.** Both the list of submitted votes and the list of authorized credentials are anonymized by applying a random permutation, implemented with a *mix network* [16]. In the mix, each tabulation teller in turn applies its own random permutation.

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

6. **Decrypt.** The remaining choices, but not credentials, are decrypted. The final tally is publicly computable.

[11] Other kinds of ballots can be encoded into one of these supported forms. For example, conditional ballots, in which a voter selects “yes” or “no” on some issue, then is offered particular candidates based on this selection, can be encoded as a plurality vote on a pair of a selection and a candidate.

[12] For example, the adversary could issue each voter a unique, large number, then demand that the voter submit that number as the voter’s choice. If that number does not appear in the final list of decrypted choices, the adversary knows that the voter did not comply.

Verifying an election. Tabulation is made publicly verifiable by requiring each tabulation teller to post proofs that it is honestly following the protocols. All tabulation tellers verify these proofs as tabulation proceeds. An honest teller refuses to continue when it discovers an invalid proof. Anyone can verify these proofs during and after tabulation, yielding universal verifiability. A voter can also verify that his vote is present in the set retrieved by the tabulation tellers, yielding voter verifiability.

4 Security Evaluation

The Civitas voting scheme requires certain assumptions about the trustworthiness of agents and system components. We discuss what attacks are possible when these trust assumptions are violated, and what defenses an implementation of the scheme could employ.

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

There must be some period of time during which the adversary cannot simulate the voter. Otherwise the system could never distinguish the adversary from the voter, so the adversary could register and vote on behalf of a voter. Registration is a good time for this assumption because it requires authentication and can be done far in advance of the election.

During registration, Civitas authenticates voters with their registration keys. So this assumption restricts the adversary from acquiring a voter's key before the voter has registered. However, voters might attempt to sell their private registration keys, or an adversary might coerce a voter into revealing the voter's key.[13] Both attacks violate Trust Assumption 1 by allowing the adversary to simulate a voter.

One possible defense would be to store private keys on tamper-resistant hardware, which could enforce digital non-transferability of the keys. This is not a completely effective defense, as voters could physically transfer the hardware to the adversary. Preventing such physical transfers is not generally possible, but they could be discouraged by introducing economic disincentives for voters who relinquish their keys. For example, the Estonian ID card, which contains private keys and is used for electronic voting, can be used to produce legally binding cryptographic signatures [77]. Voters would be unlikely to sell such cards, although coercion would remain a problem.

Another possible defense is to change authentication to use in-person registration as an alternative to private keys. Each registration teller would either be an *online* teller, meaning voters register with that teller remotely, or an *offline* teller, meaning voters must register in person with that teller. Offline registration tellers would be trusted to authenticate voters correctly, preventing the adversary from masquerading as the voter. At least one offline registration teller would need to exist in any election, ensuring that voters register in person with at least one teller.

For deployments of Civitas in which this trust assumption does not hold, we recommend requiring in-person registration. This compromises of our goal of a fully remote system. But it is a practical defense, since voting could still be done remotely, registration could be done far in advance of the actual election, and a single credential could be reused for multiple elections.[14]

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.*

[13] Note that these attacks are relevant only to registration, not voting, because the voter's registration key is not used during the voting protocol.

[14] Such reuse would require strengthening Trust Assumptions 2 and 6 to honesty of tellers across multiple elections.

Constructing a fake credential requires the voter to modify at least one of the shares received during registration. 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 corrupt the voter's client so that it records all credential shares received from tellers. Then the adversary can ask the client to reveal the plaintext credential shares corresponding to the encrypted network messages. In this scenario, the voter cannot lie to the adversary about his credential shares, meaning that the voter could now sell his credential and is no longer protected from coercion. So an untappable channel is required for distribution of at least one share. The voter must also trust the teller who issued that share not to reveal it.[15]

An untappable channel is the weakest known assumption for a coercion-resistant voting scheme [6, 25, 47, 58, 81]. Replacing this with a more practical assumption has been an open problem for at least a decade [26]. Offline registration tellers, discussed with Trust Assumption 1, could ensure an untappable channel by supervising the registration process. Our prototype of the client employs enforced erasure of all credential shares once the voter's credential is constructed, preventing the voter from reporting shares to the adversary.

Trust Assumption 3. *Voters trust their voting clients.*

Voters enter votes directly into their clients. No mechanism ensures that the client will preserve the integrity or the confidentiality of votes. A corrupt voting client could violate coercion resistance by sending the plaintext of the voter's credential and choice to the adversary. A corrupt client could also violate verifiability by modifying the voter's credential or choice before encrypting it.

Clients could be corrupted in many ways. The machine, including the network connection, could be controlled by the adversary. Any level of the software stack, from the operating system to the client application, could contain vulnerabilities or be corrupted by malicious code. The adversary might even be an insider, compromising clients during their development and distribution.

Current research aims to solve this problem by changing how voters enter their votes [17, 56, 62, 93]. The voting client is decomposed into multiple (hardware and software) components, and the voter interacts with each component to complete the voting process. For example, voting might require interacting with a smart card to obtain a randomized ballot, then interacting with a client to submit a vote on that ballot.[16] Now the voter need not trust a single client, but instead that the components implementing the client will not collude. Complementary research aims to leverage trusted computing technology [89]. For example, attestation could be used to prove that no level of the hardware or software stack has been changed from a trusted, pre-certified configuration. Integrating these kinds of defenses into Civitas is important future work.

Note that this trust assumption does not require all voters to trust a single client implementation. Rather, voters may choose which client they trust. This client could be obtained from an organization the voter trusts, such as their own political party or another social organization. These organizations are free to implement their own Civitas client software on their own hardware, and to make their source code publicly available. This freedom improves upon current direct-recording electronic (DRE) voting systems, in which voters are often forced by local election authorities to use particular proprietary (or closed-source) clients that are known to contain vulnerabilities [59,61,91].

[15] Note that a voter must know which registration teller he is trusting, which is stronger than Trust Assumptions 5 and 6.

[16] Another example is the use of paper as one of the components. However, this is incompatible with remote electronic voting.

Another advantage over DREs is that diverse clients, provided by several organizations, could reduce the incentive to attack Civitas by raising the cost of mounting an attack.

Requiring trusted voter clients compromises our goal of a remote voting system. Even if voters download a client from a trusted organization, the software stack on a voter's machine might not be trustworthy. Thus voters might need to travel to a location where an organization they trust has provided a client application running on a trustworthy hardware and software platform.

Trust Assumption 4. *The channels on which voters cast their votes are anonymous.*

Without this assumption, the adversary could observe network traffic and learn which voters have voted, trivially violating coercion resistance—although the adversary still could not learn the voter's choice or credential.

Our prototype of Civitas does not implement its own anonymous channel because the construction of trustworthy anonymous channels is an orthogonal research problem. It seems likely that existing anonymizing networks, such as Tor [33], would suffice if made sufficiently reliable.[17]

Trust Assumption 5. *At least one of the ballot boxes to which a voter submits his vote is correct.*

A *correct* ballot box returns all the votes that it accepted to all the tabulation tellers. This is weaker than the standard assumption (less than a third of the ballot boxes fail) made for Byzantine fault tolerance [15] and multi-party computation [42], which both require more expensive protocols.

Trust Assumption 6. *There exists at least one honest tabulation teller.*

If all the tellers were corrupted, then the adversary could trivially violate coercion resistance by decrypting credentials and votes. This assumption is not needed for verifiability, even if all the tellers collude or are corrupted—the proofs posted by tellers during tabulation will reveal any attempt to cheat. Fault tolerance techniques [19, 38, 82] would increase the difficulty of corrupting all the tellers.

Attacks on election authorities. Trust Assumptions 2, 5, and 6 allow all but one election authority of each kind to be corrupted. But certain attacks might still be mounted:

- A corrupt registration teller might fail to issue a valid credential share to a voter. The voter can detect this, but coercion resistance requires that the voter cannot prove that a share is valid or invalid 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 might attempt to alter messages. But this is detectable since messages are signed. A bulletin board might also delete messages. This is an attack on availability, which is addressed in Section 11.

- A corrupt registrar might add fictitious 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.

[17] A vote typically fits into just three packets, so scalability and timing attacks seem unlikely to present problems.

- A corrupt supervisor might post an incorrect ballot design, stop an election early, or even attempt to simulate an election with only one real voter. Voters and tabulation tellers should cease to participate in the election once the supervisor exhibits such behavior.

All election authorities might be simultaneously corrupted if they all run the same software. For example, an insider working at the software supplier might hide malicious code in the tabulation teller software. As discussed in Trust Assumption 6, this attack could violate coercion resistance, but it could not violate verifiability. To defend against insider attacks, election authorities should use diverse implementations of the Civitas protocols.

Trust Assumption 7. *The Decision Diffie-Hellman (DDH) and RSA assumptions hold, and SHA-256 implements a random oracle.*

DDH and RSA are standard cryptographic assumptions. The more fundamental assumption for Civitas is DDH, as the JCJ security proof is a reduction from it.

5 Cryptographic Components

Civitas uses many cryptographic components. This section gives an overview of these; Appendix B contains a detailed specification of the protocols. Many components require posting messages to the bulletin board. These messages must be signed by the poster. Also, a variety of zero-knowledge proofs are used to enforce the honest execution of protocols. These proofs are made non-interactive via the Fiat-Shamir heuristic [37], so their security is in the random oracle model [7]. Civitas implements a random oracle with SHA-256.

Security proof. The security of Civitas follows from the JCJ security proof [58] and the individual security proofs of each component, cited below. We give a security proof for the registration protocol in Appendix A.

5.1 Setup phase

Keys. The supervisor posts RSA public keys representing the election authorities. These keys are used for authentication of agents and messages. The choice of RSA is for convenience, since many real-world organizations already have RSA keys, but could be replaced by another cryptosystem. The tabulation tellers also generate a distributed El Gamal public key, described below. The registrar posts each voter's registration public key (RSA, again for convenience) and designation public key (El Gamal).

Encryption scheme. Civitas implements a distributed El Gamal scheme similar to Brandt's [11]. The supervisor posts a message (p, q, g) describing the cryptosystem parameters: a prime $p = 2kq + 1$, where q is also prime, and a generator g of the order q subgroup of $\mathbb{Z}_p^*$. This subgroup, denoted $\mathcal{M}$, is the message space of the cryptosystem. The tabulation tellers generate an El Gamal public key K_{TT} for which each teller holds a share of the corresponding private key. Encryption of message m under key K with randomness r is denoted $\mathsf{Enc}(m; r; K)$. We omit r or K from this notation when they are unimportant or clear from context. Decryption of a ciphertext c that was encrypted under key K_{TT}, denoted $\mathsf{Dec}(c)$, requires all tabulation tellers.

El Gamal encryption is *homomorphic* with respect to multiplication. That is, $\mathsf{Enc}(m) \cdot \mathsf{Enc}(n) = \mathsf{Enc}(m \cdot n)$. El Gamal permits a probabilistic *reencryption* operation, denoted $\mathsf{Reenc}(c)$ for a ciphertext c, which produces a new encryption of the same plaintext. Encryption can be made *non-malleable*, preventing the use of homomorphisms and reencryption, by the use of Schnorr signatures [84]. 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 [83]. 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 [18]. 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 construction for credentials, based on ideas found in earlier work [26, 47, 58]. The security of this construction is proved in Appendix A.

For each voter, each registration teller i individually generates a random element of $\mathcal{M}$ as private credential share s_i. The corresponding public share S_i is $\mathsf{Enc}(s_i; K_{\mathrm{TT}})$. The registration teller posts S_i on the bulletin board and stores s_i for release during registration. After all tellers have posted a share, the voter's public credential S is publicly computable as $\prod_i \mathsf{Enc}(s_i; K_{\mathrm{TT}})$, which by the homomorphic property is equal to $\mathsf{Enc}(\prod_i s_i; K_{\mathrm{TT}})$.

5.2 Voting phase

Registration. To acquire a private credential, a voter contacts each registration teller. The voter authenticates using his registration key, then establishes a shared AES session key using the Needham-Schroeder-Lowe [64] protocol. The voter requests registration teller i's share s_i of the private credential. The registration teller responds with (s_i, r, S'_i, D), where r is random, $S'_i = \mathsf{Enc}(s_i; r; K_{\mathrm{TT}})$ and D is a *designated-verifier reencryption proof* (DVRP) due to Hirt and Sako [47]. The proof shows that S'_i is a reencryption of S_i, the public credential share. Construction of this proof requires the voter's public designation key. The voter verifies that S'_i was computed correctly from s_i and r, then verifies the DVRP. These verifications convince 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 private credential s, where $s = \prod_i s_i$.

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

$$\langle \mathsf{Enc}(s; K_{\mathrm{TT}}), \mathsf{Enc}(v; K_{\mathrm{TT}}), P_w, P_k \rangle$$

to some or all of the ballot boxes, where s is the voter's private credential, v is the voter's choice, and P_w and P_k are zero-knowledge proofs. P_w, implemented with a *1-out-of-L reencryption proof* due to Hirt and Sako [47], shows that the vote is well-formed with respect to the ballot design of the election. Given $C = \{c_i \mid 1 \leq i \leq L\}$ and c, this reencryption proof shows there exists an i such that $c_i = \mathsf{Reenc}(c)$. P_k, implemented by adapting a proof due to Camenisch and Stadler [13], shows that the submitter simultaneously knows s and v. This defends against an adversary who attempts to post functions of previously cast votes.

Resisting coercion. To construct a fake credential, a voter chooses at least one registration teller and substitutes a random group element $s'_i \in \mathcal{M}$ for the share s_i that registration teller sent to the voter. The voter can construct a DVRP that causes this fake share to appear real 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 unless the adversary observed the channel used by the registration teller and voter during registration (in which case the adversary has seen the real proof). By Trust Assumption 2, there exist some teller and channel that the adversary does not control, so it is always possible for voters fake credentials.

5.3 Tabulation phase

Ballot boxes. Recall from Section 3 that ballot boxes are instances of an insert-only log service. Ballot boxes have one additional function, reporting their contents at the end of an election. When the supervisor closes the election, each ballot box posts a commitment to its contents on the bulletin board. The supervisor then posts his own signature on all these commitments, defining the set of votes to be tabulated. Thus, if a voter posts a vote to at least one correct ballot box, the vote will be tabulated.[18] Note that ballot boxes do not check validity of votes.

Since ballot boxes operate independently, never contacting other ballot boxes, this ballot box construction scales easily. Moreover, this construction ensures that all votes are available for tabulation—a requirement of universal verifiability—without expensive fault tolerance protocols.

Mix network. A mix network is used to anonymize submitted votes and authorized credentials. Civitas implements a reencryption mix network made verifiable by randomized partial checking [52], in which each teller in the network performs two permutations.[19]

Duplicate and invalid credential elimination. It would be easy to eliminate votes containing duplicate or invalid credentials if credentials could be decrypted. However, this would fail to be coercion-resistant, because voters' private credentials would be revealed. Instead, a zero-knowledge protocol called a *plaintext equivalence test* (PET) is used to compare ciphertexts. Given c and c', a PET reveals whether $\mathsf{Dec}(c) = \mathsf{Dec}(c')$, but nothing more about the plaintexts of c and c'. Civitas implements a PET protocol due to Jakobsson and Juels [51]. For duplicate elimination, a PET must be performed on each pair of submitted credentials. Similarly, to eliminate invalid credentials, PETs must be performed to compare each submitted credential with every authorized credential.[20] These pairwise tests cause credential elimination to take quadratic time.

Table 1: Modular exponentiations per block

Agent	Action	Protocol	BB
RT	Generate all credentials	$4K$	K
	Distribute all credentials	$14K$	–
Voter	Retrieve a credential	$12A$	A
	Vote	$4C+7$	–
TT	Retrieve data	–	$AK+A+1$
	Verify proofs	$4M(C+1)$	–
	Eliminate duplicates	$\binom{M}{2}(8A-1)$	$3A$
	Anonymize (mixes)	$2(A+1)(M+K)$	$2A$
	Eliminate invalids	$KM(8A-1)$	$3A$
	Decrypt	$K(4A-1)$	A

6 Scalability

There are two main challenges for scalability in Civitas. First, elimination of duplicate and invalid credentials takes quadratic time. Second, tabulation requires each teller to perform computation for each vote.

Our solution to both challenges is to group voters into *blocks*, which are virtual precincts. Like real-world precincts, the tally for each block can be computed independently, block results are public, and voters are anonymous within their block. Unlike real-world precincts, the assignment into blocks need not be based on physical location. For example, voters might be assigned to blocks in a way that is verifiably pseudorandom, reducing the risk of reprisal by the adversary against an entire block of voters. Blocking also enables the production of *early returns*, in which a fraction of blocks are tabulated to predict the outcome of the election.

Implementing blocking is straightforward. The registrar publicly assigns each voter to a block. Each submitted vote identifies, in plaintext, the block in which its credential (supposedly) resides. Vote proof P_k is extended to make this identifier non-malleable.

Without blocking, duplicate elimination requires $O(N^2)$ PETs, where N is the number of all submitted votes. With blocking, $O(BM^2)$ PETs are required, where $B = \lfloor \frac{V}{K} \rfloor$ is the number of blocks, V is the number of voters, K is the minimum number of voters per block, and M is the maximum number of votes submitted in a block. Likewise, blocking reduces invalid credential elimination from $O(VN)$ PETs to $O(BKM)$. The B factor in each of these terms is easily parallelizable, since a different set of machines can be used to implement the tabulation tellers for each block. Tabulation time then depends on M and K, but not V. Therefore performance can scale

[18] A malicious supervisor could violate this by excluding a correct ballot box. This trust in the supervisor could be eliminated by using a more expensive agreement protocol.

[19] Randomized partial checking reveals some small amount of information about these permutations. In the worst case, when all but one teller is corrupted, the size of the set within which a vote or credential is anonymous is halved. By a result of Gomułkiewicz et al. [43], the revealed information can be made statistically small by requiring each teller to perform a total of five permutations. We estimate this would increase tabulation time by at most 3%. Mix networks based on zero-knowledge proofs [40, 70] would improve anonymity at the cost of more expensive verification.

[20] The presence of invalid credentials is an information channel. For example, if there are zero invalid credentials, then no voter submitted a vote with a fake credential. The adversary could detect this from the PET results posted on the bulletin board. To eliminate this channel, each teller could post a random number of votes with invalid credentials.

independently of the number of voters.

Table 1 identifies the number of modular exponentiations performed per block by individual agents: registration tellers (RT), tabulation tellers (TT), and voters. (Tabulation time is dominated by modular exponentiations.) 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. BB exponentiations result from RSA signatures and verifications. Exponentiations are counted under the assumption that there are no duplicate votes and that no voters abstain, maximizing the number of PETs required. Parameter A describes the number of election authorities of each kind—i.e., if $A = 4$, then there are four registration tellers, four tabulation tellers, and four ballot boxes. Regardless of A, there is a single bulletin board. Table 1 assumes a plurality ballot with C candidates.

7 Implementation in Jif

Our prototype of Civitas is implemented in Jif$_E$ [20], an extension of Jif 3.0 [67,69]. Jif is a security-typed language [90] in which programs are annotated with information-flow security policies. The Jif compiler and runtime guarantee end-to-end enforcement of these polices. Information-flow policies control both the release and propagation of information, enabling the protection of both sensitive data and data derived therefrom. Information-flow policies are therefore stronger than access control policies, which control only the release of information.

Jif security policies are expressed using the *decentralized label model* [68], which allows specification of confidentiality and integrity requirements of principals. Such policies are useful for constructing systems like Civitas, in which principals need to cooperate yet are mutually distrusting. For example, if information is labeled with confidentiality policy $\texttt{RT}_1 \rightarrow \texttt{voter}_{76}$, then principal $\texttt{RT}_1$ permits principal $\texttt{voter}_{76}$ to learn the information; such a policy would be suitable for the private credential share generated by registration teller $\texttt{RT}_1$ for $\texttt{voter}_{76}$. Similarly, if information is labeled with integrity policy $\texttt{TT}_3 \leftarrow \texttt{Sup}$, then principal $\texttt{TT}_3$ requires that only principal $\texttt{Sup}$ has influenced the information; such a policy would be suitable for the ballot design, which only the supervisor may specify.

In general, a principal p may specify a set R of *readers* in confidentiality policy $p \rightarrow R$. Jif$_E$ extends Jif with *declassification* and *erasure* policies [21], which allow principals to state *conditions* on when the set of readers in a confidentiality policy may be changed.

Declassification policies allow the set of readers of information to be expanded. For example, in the implementation of mix networks, each tabulation teller must commit to random bits. The bits are then revealed and used to verify the mix. The security of the mix requires maintaining the secrecy of these bits until all tellers have committed. In the code, this requirement is expressed using a declassification policy. The policy annotating the variable storing the random bits of $\texttt{TT}_i$ indicates that the information is readable only by $\texttt{TT}_i$ until condition *AllCommitted* is satisfied, upon which the information may be declassified to be readable by all principals. *AllCommitted* becomes true at the program point where all commitments have been received.

Erasure policies mandate conditions upon which the set of readers must be restricted. For example, each registration teller must store a private credential share for each voter until the voter requests it. After this, the teller may erase the share, ensuring that the share cannot later be dis-

Table 2: Lines of Jif$_E$ code per component

Component	LOC
Tabulation teller	5,740
Common	3,173
Registration teller	1,290
Supervisor	1,138
Log service (bulletin board and ballot box)	911
Voter client	826
Registrar	308
Total	13,386

closed.[21] In the code, the variable storing the share is annotated with an erasure policy indicating that this information becomes unreadable by all principals when condition *Delivered* is satisfied. *Delivered* becomes true at the program point where receipt of the share has been acknowledged by the voter. The Jif$_E$ compiler inserts code at that point to erase the information from memory.

Our implementation of Civitas totals about 13,000 lines of Jif$_E$ code. Table 2 gives the number of lines of code in each component; common code includes shared data structures and utility methods for retrieving and caching election information. About 8,000 additional lines of Java code are used to perform I/O and to implement number-theoretic operations such as encryption and zero-knowledge proofs.

8 Ranked Voting Methods

Plurality voting, in which each voter chooses a single candidate and the winner is the candidate who receives the most votes, is the best-known voting method. However, it is subject to the spoiler effect, in which the presence of minor candidates affects the outcome. The spoiler effect is mitigated by *ranked voting methods*, in which each voter submits a (partial or total) ordering of the candidates. Ranked voting methods are used by many organizations and by several countries; examples of such methods include single transferable vote (a.k.a. instant runoff), the Condorcet family of methods, and the Borda count [10]. Ranked methods are attractive because they enable voters to express more information about their preferences, and that information can be used to produce a better collective decision. For example, Condorcet methods extend the principle of majority rule to ranked preferences: A candidate who would defeat every other candidate in one-on-one elections is declared the Condorcet winner. Civitas implements a Condorcet voting method, in addition to plurality voting and approval voting.

Coercion resistance is a challenge with ranked voting methods because ordered preferences introduce a covert channel: Voters can encode information into low-order preferences in their votes. 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 encoded, allowing voters to covertly identify themselves. Coercing voters into using this covert channel is sometimes referred

[21] Erasure is a design choice that impacts recovery from voters' accidental loss or deletion of credentials. If tellers do not erase shares, then tellers can reissue credentials. But if tellers do erase shares, then reissue is not possible. Instead, tellers would need to revoke lost shares and issue new shares. This is left as future work.

to as an *Italian attack*.

Civitas eliminates the covert channel by encoding the voter's ranking into $\binom{C}{2}$ votes, where C is the number of candidates [24]. Each vote expresses one of three preferences between candidates i and j: (1) i is preferred over j, (2) j over i, or (3) i and j are tied.[22] After votes are decrypted, these anonymized preferences are used to select the winner. This encoding requires $\binom{C}{2}$ separate elections to be tabulated, but offers coercion resistance: The adversary is unable to learn more about an individual voter's low-order preferences than can be learned from the aggregate preferences of all voters, which are revealed by the final tally.

It is also possible to eliminate the covert channel yet tabulate only a single election. Suppose a ranking were encoded as a matrix m of preferences, where each cell is either 0 or 1. If $m[i, j]$ is 1, then the voter prefers candidate i over candidate j.[23] When submitting this matrix as a choice during voting, suppose that instead of encrypting the entire matrix, the voter instead encrypts each cell individually in an additively homomorphic cryptosystem supporting reencryption, perhaps either exponential El Gamal [1] or Paillier [74]. The tabulation protocol would remain essentially unchanged until the final decryption phase. The individual matrices, still encrypted, would be summed using the homomorphic property. The resulting encrypted sum matrix, containing the aggregated voter preferences, would then be decrypted.[24] Implementation of this in Civitas is left as future work.

9 Performance

A voting system is practical only if tabulation 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.

Notions of reasonable time, cost, and security may differ depending on the election or the observer. In current U.S. elections, accurate predictions of election results are available within a few hours. Therefore, we chose a target tabulation time of five hours. The two most important parameters affecting security are K, the minimum number of voters within each block, and A, the number of authorities of each kind.[25] As reasonable values for these parameters, we chose $K = 100$ and $A = 4$. Anonymity within 100 voters seems comparable to what is available in current real-world elections, where results are tabulated at a precinct level and observers might correlate voters with ballots.[26] Similarly, four mutually distrusting authorities might offer better oversight than real-world elections.

Experiment design. We used Emulab [92] as an experiment testbed. The experiments ran on machines containing 3.0 GHz Xeon processors and 1 GB of RAM, networked on a 1 Gb LAN. Note that only tabulation tellers actually need hardware this fast, whereas voters could use substantially

[22] This encoding does not guarantee that voters submit a true order. Voters could instead submit votes containing cycles, e.g., $A > B$, $B > C$, and $C > A$. While this is still a Condorcet method—if a Condorcet winner exists, it will be elected—cyclic votes may introduce new possibilities for strategic voting. To eliminate this, voters could be required to submit additional zero-knowledge proofs, establishing that no cycles exist in their votes.

[23] This encoding also allows voters to submit votes containing cyclic preferences, so voters might again be required to submit additional zero-knowledge proofs.

[24] In general, decryption in exponential El Gamal requires taking a discrete log. But if the block size B is not overly large, decryption could be efficiently implemented with a lookup table mapping i to g^i, for $0 \leq i \leq B$.

[25] Recall from Section 6 that if $A = 4$, then there are four registration tellers, four tabulation tellers, and four ballot boxes.

[26] Random block assignment might even offer stronger anonymity than real-world elections.

less powerful hardware without impacting performance or the voting experience. Our machines ran Red Hat Linux 9.0 and Java 1.5.0_11. For RSA, AES, and SHA implementations, we used Bouncy Castle JCE provider 1.33. We implemented the remaining cryptographic functionality, including El Gamal and zero-knowledge proofs, ourselves. We used a C library, GMP 4.2.1, for implementations of modular exponentiation and multiplication.

Key lengths were chosen to meet or exceed NIST recommendations for 2011–2030 [5]. We used 128-bit AES keys, 2048-bit RSA keys, and 224-bit El Gamal keys from a 2048-bit group—i.e., $|p| = 2048$ and $|q| = 224$. A modular exponentiation in this size group required about 3.7 ms.

Each experiment simulated all phases of a complete election, including all the cryptographic protocols in Section 5. Therefore the results should be representative of a real deployment. All experiments used plurality ballots with three candidates. No voters abstained, so $N \geq V$ and $M \geq K$.[27] Experiments were repeated three times, and we report the sample mean. The sample standard deviation was always less than 2% of the mean.

Setup and voting time. Generation of keys and credentials 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. A voter client takes about 325 ms to acquire a credential share from a registration teller, and about 20 ms to submit a vote to a ballot box. Thus, for four authorities, it takes a voter less than 1.4 seconds to retrieve credentials and submit a vote. From the registration teller's perspective, it takes about 200 ms of CPU time to distribute a single voter's credential share. A registration teller could therefore process 18,000 voters per hour.

Tabulation time and space. Figure 2(a) shows the results of four tabulation tellers processing blocks sequentially, where V is a multiple of K. The data indicate that Civitas requires 39 seconds per voter per authority to tabulate a single block, and that votes from 500 voters, in blocks of 100, can be tabulated in five hours. (The time to combine the block tallies is negligible.) Parameters A and K have non-linear effects on tabulation time, as shown in Figure 2(b) and Figure 2(c). Communication increases quadratically in A, and PETs take time proportional to K^2. Figure 2(c) indicates that a block of 200 voters can be tabulated in less than five hours.

The independence of blocks can be exploited to decrease tabulation time by processing blocks in parallel. Given a set of tabulation teller machines for each block, the data in Figure 2(a) predict that tabulation could be completed in about 65 minutes, independent of V. 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.

The memory footprint of Civitas is very small. With $M = 100$, the active set of a tabulation teller is never more than 8 MB. 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 $K \leq M$). The storage space needed for the entire bulletin board is less than 620 MB for an election where $K = 100$, $V = 100$, and $A = 4$. Our prototype uses a verbose XML-like message format, so we expect that storage space requirements could be reduced significantly.[28]

[27] Recall that N is the number of votes submitted and M is the maximum number of votes submitted in a block.

[28] Note that voters do not need to download the entire bulletin board to verify inclusion of their votes. Rather, a voter would need to download only the list of votes (about 160 kB) used as input to the tabulation protocol, then check that his

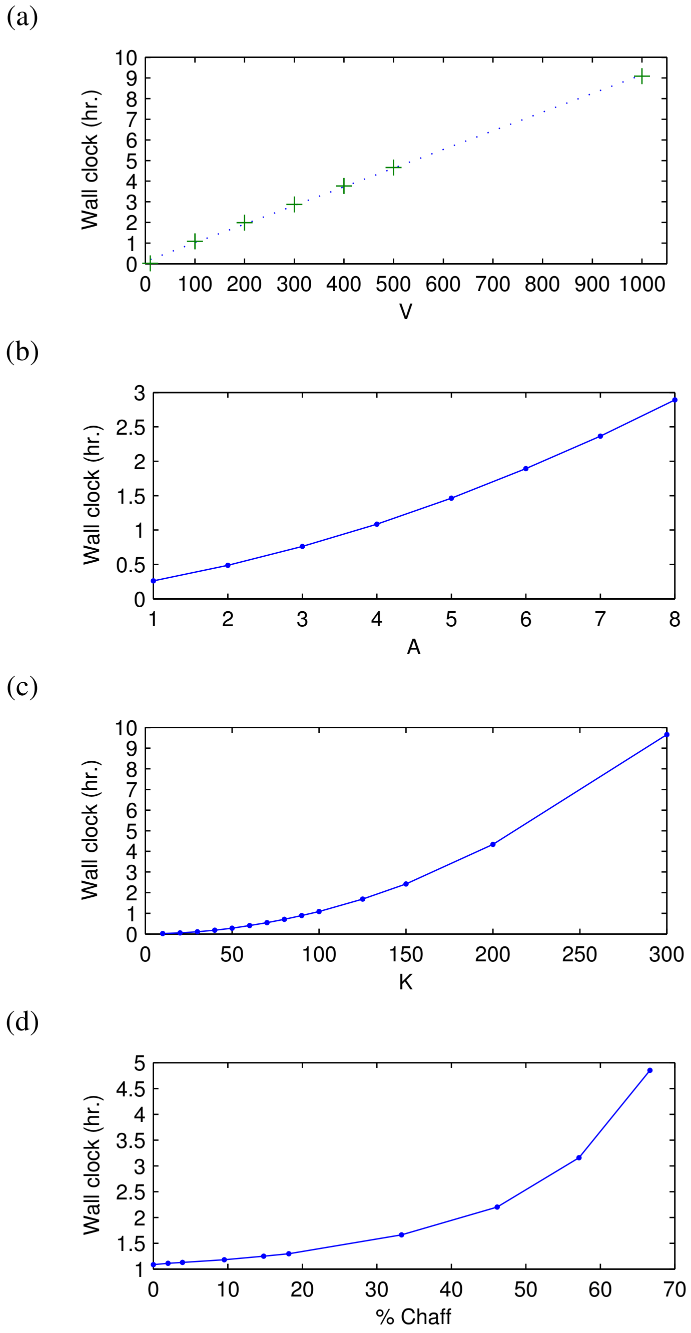

Figure 2: Tabulation time vs. (a) Voters: $K = 100, A = 4$; (b) Authorities: $K = V = 100$; (c) Anonymity: $V = K, A = 4$; (d) Chaff: $K = V = 100, A = 4$

Chaff. We refer to votes containing invalid and duplicate credentials as *chaff* because they are eliminated during tabulation. Because chaff increases the number of votes in a block, it increases tabulation time similarly to increasing anonymity parameter K. Figure 2(d) shows how tabulation time varies as a function of the percentage of chaff votes in each block. With fraction c chaff (split between invalid and duplicate credentials), there are $M = \frac{V}{1-c}$ votes in a block. All the other graphs in this study assume $c = 0$.

Cost. A government election in a stable Western democracy currently costs \$1 to \$3 per voter [49]. Civitas would increase the cost of computing equipment but could reduce the costs associated with polling places and paper ballots. A dual-core version of our experiment machines is currently available for about \$1,500, so the machine cost to tabulate votes from 500 voters in five hours (with $K = 100$ and $A = 4$) is at worst \$12 per voter, and this cost could be amortized across multiple elections. Moving to multicore CPUs would also be likely to reduce tabulation time, since tabulation is CPU-bound (utilization is about 70–85% during our experiments), has a small memory footprint, and can be split into parallel threads that interact infrequently. Costs could be reduced dramatically if trust requirements permit a tabulation teller to lease compute time from a provider.[29] One provider currently offers a rate of \$1 per CPU per hour on processors similar in performance to our experiment machines [88]. At this rate, tabulation for 500 voters would cost about 4¢ per voter—clearly in the realm of practicality.

Reducing security parameters also reduces cost. For example, halving K approximately quarters tabulation time. So for a ten-hour, $K = 50$, $A = 3$ election, the cost per voter would be about ten times smaller than a five-hour, $K = 100$, $A = 4$ election. El Gamal key lengths also have a significant impact. Figure 2(c) shows that, for 224-bit keys from a 2048-bit group, K can be as high as 200 while maintaining a tabulation time of under five hours. With 160-bit keys from a 1024-bit group (secure, according to NIST, from 2007–2010 [5]), K can be increased to 400. Using 256-bit keys from a 3072-bit group (secure until after 2030) currently requires decreasing K to 125.

Real-world estimates. In the 2004 general election for President of the United States, just under 2.3 million votes were reported by the City of New York Board of Elections [22]. Using the worst-case estimate we developed above, \$12 per voter, the one-time hardware cost for using Civitas to tabulate this election would be at most \$27.6 million. In comparison, Diebold submitted an estimate in 2006 of \$28.7 million in one-time costs to replace the city's mechanical lever voting machines with optical scan machines [30]; hardware and software costs accounted for \$10.2 million of this estimate [31]. Although we cannot make any strong comparisons, the cost of Civitas does seem to be about the same order of magnitude.

10 Related Work

Voting schemes. Cryptographic voting schemes can be divided into three categories, based on the technique used to anonymize votes. Here we cite a few examples of each type. In schemes based on *homomorphic encryption* [8, 26, 47, 80], voters submit encrypted votes that are never decrypted.

vote is in this list.

[29] Essentially, this means trusting the provider with the teller's El Gamal private key share for that election so the provider can compute decryption shares. To avoid giving the provider the key share, computation might be split between the provider and teller, with the teller computing only these decryption shares. This would result in the teller performing only about 10% of the total number of modular exponentiations.

Rather, the submitted ciphertexts are combined (using some operation that commutes with encryption) to produce a single ciphertext containing the election tally, which is then decrypted. *Blind signature* schemes [39,72,73] 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 [6,16,65,81], voters authenticate and submit encrypted votes. Votes are anonymized using a mix, and anonymized votes are then decrypted. JCJ and Civitas are both based on mix networks.

To optimize JCJ, Smith [87] proposes replacing PETs with reencryption into a deterministic, distributed cryptosystem. However, the proposed construction is insecure. The proposed encryption function is $\mathsf{Enc}(m; z) = m^z$, where z is a secret key distributed among the tellers. But to test whether s is a real 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 real with high probability. Araújo et al. [2] are studying another possible replacement for PETs, based on group signatures.

Civitas differs from JCJ in the following ways:

- JCJ assumes a single trusted registration authority; Civitas factors this into a registrar and a set of mutually distrusting registration tellers. As part of this, Civitas introduces a construction of credential shares.
- JCJ does not specify a means of distributing credentials; Civitas introduces a protocol for this and proves its security.
- JCJ has voters post votes to the bulletin board; Civitas introduces ballot boxes for vote storage.
- JCJ supports plurality voting; Civitas generalizes this to include approval and ranked voting methods.
- JCJ left many of the cryptographic components described in Section 5 unspecified (though JCJ also provided helpful suggestions for possible implementations); Civitas provides concrete instantiations of all the cryptographic components in the voting scheme.
- JCJ, as a voting scheme, did not study the scalability of tabulation or conduct experiments; Civitas, as both a scheme and a system, introduces blocking, studies its scalability, and reports experimental results.

Voting systems. To our knowledge, Civitas offers stronger coercion resistance than other implemented voting systems. Sensus [27], based on a blind signature scheme known as FOO92 [39], offers no defense against coercion. Neither does EVOX [46], also based on FOO92. Both systems allow a single malicious election authority to vote on behalf of voters who abstain. EVOX-MA [34] addresses this by distributing authority functionality. REVS [57,63] extends EVOX-MA to tolerate failure of distributed components, but does not address coercion. ElectMe [85] is based on blind signatures and claims to be coercion resistant, but it assumes the adversary cannot corrupt election authorities. If the adversary learns the ciphertext of a voter's "ticket," the scheme fails to be receipt-free. ElectMe also is not universally verifiable. Voters can verify their votes are recorded correctly, but the computation of the tally is not publicly verifiable. Adder [60] implements a homomorphic scheme in which voters authenticate to a "gatekeeper." If the adversary were to corrupt this single component, then Adder would fail to be coercion-resistant.

Kiayias [60] surveys several voting systems from the commercial world. These proprietary systems do not generally make their implementations publicly or freely available, nor do they appear to offer coercion resistance. The California top-to-bottom review [91] of commercial electronic voting systems suggests that these systems offer completely inadequate security.

The W-Voting system [62] offers limited coercion resistance. It requires voters to sign votes, which appears susceptible to attacks in which a coercer insists that the voter abstain or submit a vote prepared by the coercer. It also allows voters to submit new votes, which replace older votes. So unlike Civitas, an adversary could successfully coerce a voter by forcing the voter to submit a new vote, then keeping the voter under surveillance until the end of the election.

Prêt à Voter 2006 [79] offers a weak form of coercion resistance, if voting is supervised. The construction of ballots depends on non-uniformly distributed seeds, which might enable the adversary to learn information about how voters voted. In remote settings, Prêt à Voter offers no coercion resistance. The adversary, by observing the voter during voting, will learn what vote was cast.

VoteHere [70] offers coercion resistance, assuming a supervised voting environment. Removing this assumption seems non-trivial, since the supervised environment includes a voting device with a trusted random number generator. This generator could be subverted in a remote setting, enabling the adversary to learn the voter's vote.

The primary goal of Punchscan [76] is high integrity verification of optical scan ballots. Punchscan does not claim to provide coercion resistance. Instead, under the assumption that voting takes place in a supervised environment, Punchscan offers a weaker property: The adversary learns nothing by observing data revealed during tabulation. This assumption rules out coercion-resistant remote voting. For confidentiality, Punchscan assumes that the election authority is not corrupted, even partially, by the adversary.

Voting methods. Smith [86] proposes implementations of single transferrable vote (STV) and range voting that defend against Italian attacks. Heather [44] proposes an implementation of STV for Prêt à Voter.

11 Toward a Secure Voting System

Some open technical problems must be solved before Civitas, or a system like it, could be used to secure national elections. Two such problems are that Civitas assumes a trusted voting client, and that in practice, the best way to satisfy two of the Civitas trust assumptions is in-person registration.

We did not address availability in this work. However, the design of Civitas accommodates complementary techniques for achieving availability. To improve the availability of election authorities, they could be implemented as Byzantine fault-tolerant services [15, 75]. 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 become unresponsive or exhibit faulty behavior, such as posting invalid zero-knowledge proofs.[30] For a threshold scheme requiring k out of n tabulation tellers to participate in decryption, no more than

[30] Recovery from these faults would need to ensure that the adversary cannot exploit any partial information from aborted subphases.

$k-1$ tellers may be corrupted, otherwise coercion resistance could be violated. For availability, a new trust assumption must be added: At least k tellers do not fail.[31]

Application-level denial of service is particularly problematic, because an adversary could insert chaff to inflate tabulation time. A possible defense, in addition to standard techniques such as rate-limiting and puzzles, would be to require a *block capability* in each submitted vote. The adversary would need to learn the capability for each block, individually, to successfully inflate tabulation time for that block. Another possible defense is to weaken coercion resistance so that chaff votes could be detected without requiring PETs. These defenses are left as future work.

We have not investigated the usability of Civitas, although usability is more important than security to some voters [45]. Management of credentials is an interesting problem for the use of Civitas. Voters might find generating fake credentials, storing and distinguishing real and fake credentials (especially over a long term), and lying convincingly to an adversary to be quite difficult. Recovery of lost credentials is also an open problem.

There are open non-technical problems as well; we give three examples. First, some people believe that any use of cryptography in a voting system makes the system too opaque for the general public to accept.[32] Second, remote electronic voting requires voters to have access to computers, but not all people have such access now. Third, some real-world attacks, such as attempts to confuse or misinform voters about the dates, significance, and procedures of elections, are not characterized by formal security models. Mitigation of such attacks is important for real-world deployments, but beyond the scope of this paper.

Finally, a report on the security of a real-world remote voting system, SERVE, identifies a number of open problems in electronic voting [55]. These problems include transparency of voter clients, vulnerability of voter clients to malware, and vulnerability of the ballot boxes to denial-of-service attacks that could lead to large-scale or selective disenfranchisement. However, Civitas does address other problems raised by the report: the voter client is not a DRE, trust is distributed over a set of election authorities, voters can verify their votes are counted, spoofing of election authorities is not possible due to the use of digital signatures, vote buying is eliminated by coercion resistance, and election integrity is ensured by verifiability.

12 Conclusion

This paper describes the design, implementation, and evaluation of Civitas, a remote voting system whose underlying voting scheme is proved secure under carefully articulated trust assumptions. To our knowledge, this has not been done before. Civitas provides stronger security than previously implemented electronic voting systems. Experimental results show that cost, tabulation time, and security can be practical for real-world elections.

Civitas is based on a previously-known voting scheme, but elaborating the scheme into an implemented system led to new technical advances: a secure registration protocol and a scalable vote storage system. Civitas thus contributes to both the theory and practice of electronic voting. But perhaps the most important contribution of this work is evidence that secure electronic voting could be made possible. We are optimistic about the future of electronic voting systems constructed, like

[31] The adversary could increase tabulation time by forcing at most $n-k$ restarts. But as long as no more than $k-1$ tellers are corrupted, the adversary cannot successfully cause tabulation to be aborted.

[32] Our stance is that it is unnecessary to convince the general public directly. Rather, we need to convince experts by using principled techniques that put security on firm mathematical foundations.

Civitas, using principled techniques.

Website

The accompanying technical report and prototype source code are available from:
`http://www.cs.cornell.edu/projects/civitas`

Acknowledgments

We thank Michael George, Anil Nerode, Nathaniel Nystrom, Tom Roeder, Peter Ryan, Fred B. Schneider, Tyler Steele, Hakim Weatherspoon, Lantian Zheng, and Lidong Zhou for discussions about this work; Jed Liu, Tudor Marian, and Tom Roeder for consultation on performance experiments; and the anonymous reviewers for their comments. We thank the participants of FEE'05 and Frontiers of Electronic Voting (Dagstuhl Seminar 07311) for feedback on preliminary versions of this work.

References

[1] Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, MIT, Aug. 2006.

[2] Roberto Araújo, Sébastien Foulle, and Jacques Traoré. On coercion-resistant schemes with linear work. In *Proc. of Frontiers of Electronic Voting: Dagstuhl Seminar 07311*, July 2007.

[3] Association for Computing Machinery. SIG elections. `http://www.acm.org/sigs/elections`, 2007.

[4] 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, Jan. 2004.

[5] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management. NIST Special Publication 800-57 Part 1, Mar. 2007.

[6] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Guillaume Poupard, and Jacques Stern. Practical multi-candidate election system. In *Proc. of ACM Symposium on Principles of Distributed Computing*, pages 274–283, Aug. 2001.

[7] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. of ACM Conference on Computer and Communications Security*, pages 62–73. ACM, Nov. 1993.

[8] Josh Daniel Cohen Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, Sept. 1987.

[9] Dan Boneh, Antoine Joux, and Phong Q. Nguyen. Why textbook ElGamal and RSA encryption are insecure. In *Proc. of International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 30–43, Dec. 2000.

[10] Steven J. Brams and Peter C. Fishburn. Voting procedures. In Kenneth J. Arrow, Amartya K. Sen, and Kotaro Suzumura, editors, *Handbook of Social Choice and Welfare*, volume 1, chapter 4. Elsevier, 2002.

[11] Felix Brandt. Efficient cryptographic protocol design based on distributed El Gamal encryption. In *Proc. of International Conference on Information Security and Cryptology*, pages 32–47, Dec. 2005.

[12] Brennan Center for Justice. The machinery of democracy: Voting system security, accessibility, usability, and cost. New York University, Oct. 2006.

[13] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Proc. of International Cryptology Conference (CRYPTO)*, pages 410–424, Aug. 1997.

[14] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *Proc. of International Cryptology Conference (CRYPTO)*, pages 90–104, Aug. 1997.

[15] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proc. of ACM Symposium on Operating System Design and Implementation*, pages 173–186, Feb. 1999.

[16] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[17] David Chaum. SureVote. `http://www.surevote.com`, 2007. International patent WO 01/55940 A1, 02 August 2001.

[18] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Proc. of International Cryptology Conference (CRYPTO)*, pages 89–105, Aug. 1992.

[19] 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.

[20] Stephen Chong and Andrew C. Myers. End-to-end enforcement of erasure. In submission.

[21] Stephen Chong and Andrew C. Myers. Language-based information erasure. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 241–254, June 2005.

[22] City of New York Board of Elections. General election results. `http://www.vote.nyc.ny.us/results.html`, 2004.

[23] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure remote voting system. Technical Report 2007-2081, Cornell University, May 2007. Revised Mar. 2008. `http://hdl.handle.net/1813/7875`.

[24] Michael R. Clarkson and Andrew C. Myers. Coercion-resistant remote voting using decryption mixes. In *Proc. of Workshop on Frontiers in Electronic Elections*, Sept. 2005.

[25] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. In *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 72–83, May 1996.

[26] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 103–118, May 1997.

[27] Laurie Faith Cranor and Ron K. Cytron. Sensus: A security-conscious electronic polling system for the Internet. In *Proc. of IEEE Hawaii International Conference on Systems Science*, pages 561–570, Jan. 1997.

[28] Debian Project. Debian vote engine (devotee). `http://www.debian.org/vote`, 2007.

[29] Stephanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proc. of IEEE Computer Security Foundations Workshop*, pages 28–42, July 2006.

[30] Diebold Election Systems. New York City BOE voting system, Cost response: Cost proposal summary, October 20, 2006. `http://www.vote.nyc.ny.us/rfi.html`.

[31] Diebold Election Systems. New York City BOE voting system, Cost response: Lever replacement solution: Optical scan pollsite system, October 20, 2006. `http://www.vote.nyc.ny.us/rfi.html`.

[32] 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, Aug. 2003.

[33] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *Proc. of USENIX Security Symposium*, pages 303–320, Aug. 2004.

[34] Brandon William DuRette. Multiple administrators for electronic voting. Bachelor's Thesis, Massachusetts Institute of Technology, 1999.

[35] Estonian National Electoral Committee. Election website. `http://www.vvk.ee/engindex.html`, 2008.

[36] David Evans and Nathanael Paul. Election security: Perception and reality. *IEEE Security & Privacy*, 2(1):24–31, Jan. 2004.

[37] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proc. of International Cryptology Conference (CRYPTO)*, pages 186–194, Aug. 1986.

[38] Stephanie Forrest, Anil Somayaji, and David Ackley. Building diverse computer systems. In *Proc. of IEEE Workshop on Hot Topics in Operating Systems*, May 1997.

[39] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 244–251, May 1992.

[40] 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.

[41] 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.

[42] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. of ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[43] Marcin Gomułkiewicz, Marek Klonowski, and Mirosław Kutyłowski. Rapid mixing and security of Chaum's visual electronic voting. In *Proc. of European Symposium on Research in Computer Security*, pages 132–145, 2003.

[44] James Heather. Implementing STV securely in prêt à Voter. In *Proc. of IEEE Symposium on Computer Security Foundations*, pages 157–169, July 2007.

[45] Paul S. Herrnson, Richard G. Niemi, Michael J. Hanmer, Benjamin B. Bederson, and Frederick C. Conrad. *Voting Technology: The Not-So-Simple Act of Casting a Ballot.* Brookings Institution Press, 2008.

[46] Mark Herschberg. Secure electronic voting over the world wide web. Master's thesis, Massachusetts Institute of Technology, 1997.

[47] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 539–556, May 2000.

[48] IEEE. Annual election website. `http://www.ieee.org/elections`, 2007.

[49] International Foundation for Election Systems. Getting to the CORE—A global survey on the cost of registration and elections, June 2006. `http://www.undp.org/governance/docs/Elections-Pub-Core.pdf`.

[50] International Organization for Standardization. Information technology—Security techniques—Key management—Part 3: Mechanisms using asymmetric techniques, 1999.

[51] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In *Proc. of International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 162–177, Dec. 2000.

[52] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proc. of USENIX Security Symposium*, pages 339–353, Aug. 2002.

[53] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 143–154, May 1996.

[54] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. Analyzing Internet voting security. *Communications of the ACM*, 47(10):59–64, Oct. 2004.

[55] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. A security analysis of the secure electronic registration and voting experiment (SERVE). `http://www.servesecurityreport.org/paper.pdf`, Jan. 2004.

[56] Rui Joaquim and Carlos Ribeiro. CodeVoting: Protecting against malicious vote manipulation at the voter's PC. In *Proc. of Frontiers of Electronic Voting: Dagstuhl Seminar 07311*, July 2007.

[57] Rui Joaquim, André Zúquete, and Paulo Ferreira. REVS—A robust electronic voting system. In *Proc. of IADIS International Conference on e-Society*, June 2003.

[58] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proc. of Workshop on Privacy in the Electronic Society*, pages 61–70, Nov. 2005.

[59] Chris Karlof, Naveen Sastry, and David Wagner. Cryptographic voting protocols: A systems perspective. In *Proc. of USENIX Security Symposium*, 2005.

[60] Aggelos Kiayias, Michael Korman, and David Walluck. An Internet voting system supporting user privacy. In *Proc. of Annual Computer Security Applications Conference*, pages 165–174. IEEE Computer Society, Dec. 2006.

[61] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic

voting system. In *Proc. of IEEE Symposium on Security and Privacy*, pages 27–42, May 2004.

[62] Mirosław Kutyłowski, Filip Zagórski, et al. W-Voting system. `http://w-vote.im.pwr.wroc.pl`, 2007.

[63] Ricardo Lebre, Rui Joaquim, André Zúquete, and Paulo Ferreira. Internet voting: Improving resistance to malicious servers in REVS. In *Proc. of IADIS International Conference on Applied Computing*, Mar. 2004.

[64] Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–136, Nov. 1995.

[65] Emmanouil Magkos, Mike Burmester, and Vassilis Chrissikopoulos. Receipt-freeness in large-scale elections without untappable channels. In *Proc. of IFIP Conference on E-Commerce, E-Business, E-Government*, pages 683–694, Oct. 2001.

[66] Rebecca Mercuri. Statement on electronic voting. `http://www.notablesoftware.com/RMstatement.html`, 2007.

[67] Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. of ACM Symposium on Principles of Programming Languages*, pages 228–241, Jan. 1999.

[68] Andrew C. Myers and Barbara Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4):410–442, Oct. 2000.

[69] Andrew C. Myers, Lantian Zheng, Steve Zdancewic, Stephen Chong, and Nathaniel Nystrom. Jif: Java information flow (software release). `http://www.cs.cornell.edu/jif`, July 2001.

[70] C. Andrew Neff. Verifiable mixing (shuffling) of ElGamal pairs. `http://www.votehere.org/vhti/documentation/egshuf-2.0.3638.pdf`, Apr. 2004.

[71] NIST. Digital signature standard (dss), Jan. 2000. FIPS 186-2.

[72] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. An improvement on a practical secret voting scheme. In *Proc. of Information Security Workshop*, pages 225–234, Nov. 1999.

[73] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Proc. of Security Protocols Workshop*, pages 25–35, Apr. 1997.

[74] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 223–238, May 1999.

[75] R. A. Peters. A secure bulletin board. Master's thesis, Technische Universiteit Eindhoven, June 2005.

[76] Stefan Popoveniuc and Ben Hosp. An introduction to Punchscan. In *Proc. of Workshop on Trustworthy Elections*, June 2006.

[77] Republic of Estonia. Digital signatures act (digitaalallkirja seadus). `http://www.riigiteataja.ee/ert/act.jsp?id=694375`, 2000.

[78] Aviel D. Rubin. Security considerations for remote electronic voting. *Communications of the ACM*, 45(12):39–44, Dec. 2002.

[79] Peter Y. A. Ryan and Steve A. Schneider. Prêt à Voter with re-encryption mixes. In *Proc. of European Symposium on Research in Computer Security*, Sept. 2006.

[80] Kazue Sako and Joe Kilian. Secure voting using partially compatible homomorphisms. In *Proc. of International Cryptology Conference (CRYPTO)*, pages 411–424, Aug. 1994.

[81] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme—A practical solution to the implementation of a voting booth. In *Proc. of International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 393–403, May 1995.

[82] Fred B. Schneider and Lidong Zhou. Distributed trust: Supporting fault-tolerance and attack-tolerance. Technical Report 2004-1924, Cornell University, Jan. 2004.

[83] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[84] Claus-Peter Schnorr and Markus Jakobsson. Security of signed ElGamal encryption. In *Proc. of International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 73–89, Dec. 2000.

[85] Anna M. Shubina and Sean W. Smith. Design and prototype of a coercion-resistant, voter verifiable electronic voting system. In *Proc. of Conference on Privacy, Security and Trust*, pages 29–39, Oct. 2004.

[86] Warren D. Smith. Cryptographic election protocols for reweighted range voting and reweighted transferable vote voting. `http://www.math.temple.edu/~wds/homepage/crirv.pdf`, Sept. 2005.

[87] Warren D. Smith. New cryptographic election protocol with best-known theoretical properties. In *Proc. of Workshop on Frontiers in Electronic Elections*, Sept. 2005.

[88] Sun Microsystems. Sun grid compute utility: Users guide. `http://www.sun.com/service/sungrid/SunGridUG.pdf`, Mar. 2007.

[89] Melanie Volkamer, Ammar Alkassar, Ahmad-Reza Sadeghi, and Stefan Schulz. Enabling the application of open systems like PCs for online voting. In *Proc. of Workshop on Frontiers in Electronic Elections*, 2006.

[90] Dennis Volpano and Geoffrey Smith. A type-based approach to program security. In *Proc. of International Joint Conference on the Theory and Practice of Software Development*, pages 607–621, Apr. 1997.

[91] David Wagner and Matthew Bishop. Voting systems top-to-bottom review. `http://www.sos.ca.gov/elections/elections_vsr.htm`, 2007.

[92] 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 *Proc. of ACM Symposium on Operating System Design and Implementation*, pages 255–270, Dec. 2002.

[93] André Zúquete, Carlos Costa, and Miguel Romão. An intrusion-tolerant e-voting client system. In *Proc. of Workshop on Recent Advances in Intrusion-Tolerant Systems*, pages 23–27, Mar. 2007.

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 real credentials.

(These requirements were extracted from steps 2 and 5 of the JCJ simulation proof of coercion resistance.) We prove below that Civitas satisfies these requirements.

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. The proofs below could be extended without difficulty to include multiple honest tellers.

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CCA2-secure (i.e., non-malleable) public-key encryption scheme. Let $\mathcal{M}$ be the message space and $\mathcal{O}$ be the space of random coins. Let $\approx$ denote computational indistinguishability and $\leftarrow$ denote uniformly random sampling.

Let $s_C \in \mathcal{M}$ be the private credential share generated for a voter by RT_C, and similarly for s_H. Then the voter's private credential is $s = s_H \cdot s_C$. Let the voter's public credential shares be S_C and S_H, and public credential be S.

A.1 Uniformly distributed credentials

To generate a credential share, an honest registration teller chooses a uniformly random element s_H from $\mathcal{M}$ and posts $S_H = \mathsf{Enc}(s_H)$ on the bulletin board. A corrupt registration teller might then attempt to use S_H to force the credential to be non-uniformly distributed. This is described by the following experiment:

```
$GenExp(n, b) =$
    $K_{\mathrm{TT}}, k_{\mathrm{TT}} \leftarrow \mathsf{Gen}(1^n);$
    $s_H \leftarrow \mathcal{M};\ S_H \leftarrow \mathsf{Enc}(s_H);$
    $S_C \leftarrow \mathrm{RT}_C(K_{\mathrm{TT}}, S_H);\ s_c \leftarrow \mathsf{Dec}_{k_{\mathrm{TT}}}(S_C);$
    $s = s_H \cdot s_C;$
    $u \leftarrow \mathcal{M};$
    if $b$ then
        output $s$
    else
        output $u$.
```

If $b = 1$, the credential generated by the tellers is output, otherwise a uniformly random credential is output. We want to show that, to an adversary, these two cases are indistinguishable.

Intuitively, this is the case because the encryption scheme is non-malleable: given S_H, the corrupt teller cannot construct any function of s_H. However, two additional technical conditions, described below, need to be met to employ this argument. To establish that these conditions hold, after both public credential shares are posted on the bulletin board, the honest teller uses an additional publicly-computable algorithm Ver to audit the share posted by the other teller. This algorithm verifies that:

1. Share S_C is a member of the ciphertext space, and
2. $S_H \neq S_C$—i.e., that RT_C did not simply copy the share posted by RT_H.

If verification fails, the honest teller reports the other teller as corrupt and refuses to proceed. Define a *compliant* teller as one that, on input S_H, outputs only S_C such that $\mathsf{Ver}(S_H, S_C)$ succeeds.

The following lemma says that if verification succeeds, then private credentials are uniformly distributed.

Lemma 1. *For all PPT compliant RT_C, $\{GenExp(n,0)\}_{n\in\mathbb{N}} \approx \{GenExp(n,1)\}_{n\in\mathbb{N}}$.*

Proof. Define three hybrids:

$$\begin{array}{lcllcl} H_0 & = & \{s_C \cdot s_H\} & & = & GenExp(n,1) \\ H_1 & = & \{r \leftarrow \mathcal{M} :\ s_C \cdot r\} & & & \\ H_2 & = & \{u\} & & = & GenExp(n,0). \end{array}$$

To show that $H_0 \approx H_1$, assume for contradiction that there exists a distinguisher D that has some non-negligible advantage in distinguishing H_0 and H_1. Using D, we can construct a machine A that breaks the CCA2-security of the encryption scheme, as follows. Machine A is given K_{TT}, a decryption oracle for k_{TT}, challenges m_0 and m_1, and a ciphertext c that encrypts one of the challenges. It then constructs instances of H_0 and H_1 where $S_H = c$, $s_H = m_0$, and $r = m_1$.

Note that constructing these instances requires A to use its decryption oracle, after receiving challenge c, to compute $\mathsf{Dec}(S_C)$. By the definition of CCA2, the oracle will not reply to this request if $S_C = c$, which is equivalent to $S_C = S_H$. However, in this case A need not succeed, because verification will fail. This means that RT_C is not compliant, making the antecedent of the lemma become false.

Machine A then asks D to distinguish the two instances. The response of D indicates whether c is an encryption of m_0 or m_1. Machine A therefore has a non-negligible advantage in breaking CCA-2 security, but this is a contradiction. So D cannot exist, and we have $H_0 \approx H_1$.

To show that $H_1 \approx H_2$, note that s_C is independent of r. Thus, since $(\mathcal{M}, \cdot)$ is a group and r is uniformly distributed, $s_C \cdot r$ is uniformly distributed, as is u.

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

The above experiment and lemma consider generating credentials for only a single voter at a time. However, credentials for many voters may be generated concurrently. In this case, we extend Ver to prevent copying shares across voters.

A.2 Indistinguishability of fake and real credentials

Let $(\mathsf{DVRP}, \widetilde{\mathsf{DVRP}})$ be a designated-verifier reencryption proof (DVRP) [47]. 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; definitions can be found in Jakobsson et al. [53].

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 $S = \mathsf{Enc}(s; r; K_{\mathrm{TT}})$ is the public share posted on the bulletin board, $S' = \mathsf{Enc}(s; r'; K_{\mathrm{TT}})$, 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:

```
$RegExp(n, b) =$
    $K_{\mathrm{TT}}, k_{\mathrm{TT}} \leftarrow \mathsf{Gen}(1^n);$
    $K_V, k_V \leftarrow \mathsf{Gen}(1^n);$
    $s \leftarrow \mathcal{M};\ r \leftarrow \mathcal{O};\ r' \leftarrow \mathcal{O};$
    $S \leftarrow \mathsf{Enc}(s; r; K_{\mathrm{TT}});\ S' \leftarrow \mathsf{Enc}(s; r'; K_{\mathrm{TT}});$
    $P \leftarrow \mathsf{DVRP}(K_V, S, S'; w);$
    $\tilde{s} \leftarrow \mathcal{M};\ \tilde{r} \leftarrow \mathcal{O};$
    $\tilde{S} \leftarrow \mathsf{Enc}(\tilde{s};\ \tilde{r}; K_{\mathrm{TT}});$
    $\tilde{P} \leftarrow \widetilde{\mathsf{DVRP}}(K_V, S, \tilde{S}; k_V);$
    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. $\{RegExp(n, 0)\}_{n \in \mathbb{N}} \approx \{RegExp(n, 1)\}_{n \in \mathbb{N}}$

Proof. Define three hybrids:

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

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 view of the adversary when the voter presents a real credential and *FakeCred* be the view from a fake credential, where

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

Indistinguishability of real 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 a multiplicative group of integers modulo a prime p, where $p = 2kq + 1$ and q is also prime. The message space $\mathcal{M}$ of this scheme is the order q subgroup of $\mathbb{Z}_p^*$. The plaintext space is $\mathbb{Z}_q$. Plaintexts must be encoded into the message space because El Gamal encryption leaks the (extended) Legendre symbol of the plaintext [9].

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 [41]; non-malleable, Schnorr and Jakobsson [84]; and distributed, Brandt [11].

ALGORITHM: El Gamal Parameter Generation

Due to: Our adaptation of the DSA parameter generation algorithm [71]
Input: Security parameters l and k
Output: Parameters (p, q, g), where $|p| = l$ and $|q| = k$

1. Select a random k-bit prime q

2. Select a random l-bit number p. Round $p - 1$ down to a multiple of $2q$ by setting p equal to $p - (p \bmod 2q) + 1$. Unless p is prime and $|p| = l$, repeat. After $2^{\log_2(l)+2}$ tries, start over by picking a new q.

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

4. 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: Encode

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

1. $M = g^m \bmod p$
```

ALGORITHM: El Gamal Encryption

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

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

We write $\mathsf{Enc}(m; r; y)$ to denote the output of Enc for a given, rather than randomly sampled, choice of r. We also write $\mathsf{Enc}^{-}(M; y)$ to denote the output of Enc when the Encode step is skipped. This is only done when M is already guaranteed to be in message space $\mathcal{M}$. Note that the main body of the paper applies encryption only to to elements of $\mathcal{M}$. Thus, $\mathsf{Enc}(\cdot)$ in the main body is equivalent to $\mathsf{Enc}^{-}(\cdot)$ in this appendix.

ALGORITHM: El Gamal Reencryption

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

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

As with encryption, we write $\mathsf{Reenc}(c; r)$ to denote output of Reenc for a given r.

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 $M$. Note that $M$ has not been decoded to an element in $\mathbb{Z}_q^*$. Doing so is apparently
   an open problem, unless $p$ is a safe prime (i.e., $k = 1$)—in which case, the Legendre symbol
   can be used for encoding and decoding.
```

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

El Gamal encryption is made non-malleable here by the use of a Schnorr signature (c, d) on the ciphertext (a, b).

ALGORITHM: Non-Malleable El Gamal Decryption

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

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

ALGORITHM: Credential Encryption

```
Input:   Public key $K_{\mathrm{TT}}$, private credential share $s \in \mathcal{M}$,
         Randomization factor $r \in \mathbb{Z}_q^*$,
         Identifiers of registration teller, $\mathit{rid}$, and voter, $\mathit{vid}$
Output:  $\mathsf{CredEnc}(s; r; K_{\mathrm{TT}}; \mathit{rid}, \mathit{vid})$

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

Credential encryption is a specialization of non-malleable encryption: the credential need not be encoded into $\mathcal{M}$, the randomization factor is provided, and the teller and voter identifiers are included in the hash used to construct the challenge for the Schnorr signature on the ciphertext.

ALGORITHM: Credential Verification

Input: Public credential share $S = (a, b, c, d)$
Identifiers of registration teller, rid, and voter, vid
Output: $\mathsf{CredVer}(S; rid, vid)$

1. $V = hash(g^d a^{-c} \bmod p, a, b, rid, vid) \bmod q$
2. Verify $V = c$

Credential verification CredVer implements function Ver used in the proof of uniformly distributed credential generation.

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 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 M. See above note on El Gamal Decryption.

B.2 Zero-knowledge proofs

PROTOCOL: KnowDlog

Due to:	Schnorr [83]
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 [18]
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 [47]
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 [47]
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:
   - $\alpha, \beta, \tilde{u} \leftarrow \mathbb{Z}_q$
   - $\tilde{a} = g^{\tilde{u}}/(\tilde{x}/x)^{\alpha} \bmod p$
   - $\tilde{b} = h^{\tilde{u}}/(\tilde{y}/y)^{\alpha} \bmod p$
   - $\tilde{s} = g^{\beta} \bmod p$
   - $\tilde{c} = hash(\tilde{E}, \tilde{a}, \tilde{b}, \tilde{s}) \bmod q$
   - $\tilde{w} = (\alpha - \tilde{c}) \bmod q$
   - $\tilde{r} = (\beta - \tilde{w})/(z_P) \bmod q$
2. $P \to V$: $\tilde{c}, \tilde{w}, \tilde{r}, \tilde{u}$
3. V: $\tilde{c}, \tilde{w}, \tilde{r}, \tilde{u}$ will verify as a DVRP, as above.

PROTOCOL: ReencPf

Due to:	Hirt and Sako [47]
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):	$t \in [1..L]$
	$r \in \mathbb{Z}_q$ 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:

     $a_i = (\frac{u_i}{u})^{d_i} g^{r_i} \bmod p$

     $b_i = (\frac{v_i}{v})^{d_i} y^{r_i} \bmod p$
   - $E = u, v, u_1, \ldots, u_L, v_1, \ldots, v_L$
   - $c = \mathit{hash}(E, a_1, \ldots, a_L, b_1, \ldots, b_L) \bmod q$
   - $w = (-rd_t + r_t) \bmod q$
   - $D = c - (\sum_{i \in [1..t-1, t+1..L]} d_i) \bmod q$
   - $R = (w + rd'_t) \bmod q$
   - $d_i^v = \begin{cases} d_i & : \quad i \neq t \\ D & : \quad i = t \end{cases}$
   - $r_i^v = \begin{cases} r_i & : \quad i \neq t \\ R & : \quad i = t \end{cases}$
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:

     $a_i^v = (\frac{u_i}{u})^{d_i^v} g^{r_i^v} \bmod p$

     $b_i^v = (\frac{v_i}{v})^{d_i^v} y^{r_i^v} \bmod p$
   - $c' = \mathit{hash}(E, a_1^v, \ldots, a_L^v, b_1^v, \ldots, b_L^v) \bmod q$
   - $D' = \sum_{i \in [1..L]} d_i^v \bmod q$
4. V: Verify $c' = D'$

PROTOCOL: VotePf

Due to:	Camenisch and Stadler [13]
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 [51]
Principals:	Tabulation tellers TT_i
Public input:	$c_j = \mathsf{Enc}(m_j; K_{\mathrm{TT}}) = (a_j, b_j)$ 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 [52]
Input:	List L of ciphertexts $[c_1, \ldots, c_m]$
	Verification direction $dir \in \{in, out\}$
Output:	RPC reencryption mix of L

1. $\pi \leftarrow$ Space of permutations over m elements
2. If $dir = in$ 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 [52]
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 [64] (steps 1–8) Our adaptation of ideas from [26, 47, 58] (steps 9–11)
Principals:	Registration teller RT_i and Voter V
Public input:	K_{TT}, K_{RT_i} Voter's El Gamal public designation key K_{V_E} Voter's RSA public registration key K_{V_R} Identifiers of election (eid), voter (vid), teller (rid), and block (bid) Public credential $S_i = \mathsf{CredEnc}(s_i; r; K_{\mathrm{TT}}; rid, mathitvid)$
Private input (RT_i):	Private credential $s_i \in \mathcal{M}$ and encryption factor $r \in \mathbb{Z}_q^*$
Private input (V):	RSA private registration key k_{V_R}

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 block bid in election eid, and that for all j, S_j is available and $\mathsf{CredVer}(S_j; rid_j, vid)$ succeeds.
4. RT_i: $N_R \leftarrow \mathcal{N}$; $k \leftarrow \mathsf{Gen}_{\mathrm{AES}}(1^l)$
5. $\mathrm{RT}_i \rightarrow V$: $\mathsf{Enc}_{\mathrm{RSA}}(rid, N_R, N_V, k; K_{V_R})$
6. V: Verify rid and N_V
7. $V \rightarrow \mathrm{RT}_i$: N_R
8. RT_i: Verify N_R
9. RT_i: $r' \leftarrow \mathbb{Z}_q^*$; $w = r' - r$; $S_i' = \mathsf{Enc}_{\mathrm{EG}}^-(s_i; r'; K_{\mathrm{TT}})$
10. $\mathrm{RT}_i \rightarrow V$: $\mathsf{Enc}_{\mathrm{AES}}(s_i, r', \mathsf{DVRP}(K_{V_E}, S_i, S_i'; w), bid); k)$
11. V: Verify $S_i' = \mathsf{Enc}_{\mathrm{EG}}^-(s_i; r'; K_{\mathrm{TT}})$, and verify DVRP against S_i from bulletin board

In Register, $\mathcal{N}$ is the space of nonces and l is the security parameter for AES.

To register, voter V completes 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.

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 even a construction of deniable encryption [14]. Register does not use non-malleable encryption in constructing S_i' because the registration teller sends plaintext s_i along with S_i'.

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 [50].

ALGORITHM: Fake credential

Due to: Our adaptation of ideas from [26, 47, 58]
Input: Private credential shares s_i, public credential shares S_i, encryption factors r_i, and DVRPs D_i from each registration teller RT$_i$ obtained from $\mathsf{Register}$
Index set L of registration tellers for which to fake shares
Voter's designation key pair (K_{V_E}, k_{V_E})
Output: Fake private credential shares $\tilde{s}_i$, fake encryption factors $\tilde{r}_i$, and fake DVRPs $\tilde{D}_i$ for each registration teller RT$_i$

1. For all $i \in L$:
    - $\tilde{r}_i \leftarrow \mathbb{Z}_q^*$
    - $\tilde{s}_i \leftarrow \mathcal{M}$
    - $\tilde{S}_i = \mathsf{Enc}^-(\tilde{s}_i; \tilde{r}_i; K_{\text{TT}})$
    - $\tilde{D}_i = \widetilde{\mathsf{DVRP}}(K_{V_E}, S_i, \tilde{S}_i; k_{V_E})$
2. For all $i \notin L$:
    - $\tilde{r}_i = r_i$
    - $\tilde{s}_i = s_i$
    - $\tilde{S}_i = \mathsf{Enc}^-(s_i; r_i; K_{\text{TT}})$
    - $\tilde{D}_i = \widetilde{\mathsf{DVRP}}(K_{V_E}, S_i, \tilde{S}_i; k_{V_E})$
3. Output $(\tilde{s}_i, \tilde{r}_i, \tilde{D}_i)$ for all i

PROTOCOL: Vote

Due to: Juels, Catalano, and Jakobsson [58]
Principals: Voter V, ballot boxes
Public input: K_{TT}, vote context ctx (including at least the election id and the block id)
Well-known choice ciphertext list $C = (c_1, \ldots, c_L)$
Private input (V): Private credential s and candidate choice c_t for some t

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

This voting protocol refines the protocol described in Section 5 in one important way: The encrypted choice the voter submits is actually a reencryption of a well-known ciphertext representing that choice. The supervisor posts these well-known ciphertexts on the bulletin board during the setup phase. The ciphertext for choice i is represented by well-known ciphertext $\mathsf{Enc}(i; 0)$. Vote does not use non-malleable encryption in constructing es because VotePf proves knowledge of plaintext s.

PROTOCOL: Tabulate

Due to:	Juels, Catalano, and Jakobsson [58]
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
Note:	This protocol is carried out for each block

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

2. Sup: Post signed copy 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 VotePfs and $\mathsf{ReencPf}$s 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