Surveying and Improving Electronic Voting Schemes

Jonathan Goulet
Jeffrey Zitelli
Advisor: Dr. Sampath Kannan

Abstract:

While recent controversies concerning the weaknesses of current voting machines have drawn some interest to methods of online voting, as well as other voting technologies, the possibility of holding large scale public elections online is only one of the motivations for studying the subject. In addition to being a candidate technology for use in elections, the problems associated with online voting provide a field on which to explore cryptographic theory. We approach the task of designing and implementing an online voting protocol with both of these motivations in mind.

Various schemes for conducting online elections have been proposed by cryptography researchers since the 1980s. These proposals have attempted to maintain the desired features of a "traditional" election in a domain where votes are submitted electronically over some public network, such as the Internet, while also seeking to expose unique advantages of online voting over other methods. We have provided an examination of these existing protocols and identified the strengths and weaknesses of each under different election premises. In the process of doing so, we have gained an understanding of the cryptographic techniques employed in these systems, and also some insights that have allowed us to make improvements to existing protocols and explore ways of making them more secure and efficient.

We have constructed an implementation of an online voting system using an adaptation of an already existing protocol. This system is comprised of an interface for conducting elections as well as a voting protocol that incorporates cryptographic methods to prevent many attacks involving adversaries that may eavesdrop or create messages, or corrupt election participants; however, security issues that are not addressed in other electronic voting protocols (such as denial of service attacks or untrusted software/hardware) were considered beyond the scope of our project. We plan to demonstrate the functionality and security of our voting system by designing a simulation of an election scenario.

Related Work:

There have been many systems proposed to solve the problem of holding local and national elections through an online system. These systems begin with the goal of implementing voting processes that employ the accessibility of the Internet, in hopes of increasing voter turnout and making it easier for citizens to vote, while at the same time maintaining the safety and security that are necessary for elections in a democratic nation. Many have begun by outlining the various features that must be present in a free and fair election, and those that are present in the current voting system used in the United States. For example, Karro and Wang require that the following criteria be met for an online voting system to be "secure and practical for large-scale elections": "Democracy", "Accuracy", "Privacy", "Verifiability", "Simplicity", "Mobility", "Efficiency", "Scalability", and "Responsibility" [*1*]. Many other proposals outline a similar list of required features. For example, the protocol proposed by Fujioka, Okamoto, and Ohta [*3*] also describes "Soundness," "Eligibility," and "Fairness" as desired properties of a good election scheme. More recently, schemes such as that of Juels and Jakobsson [*4*] also point out that "Receipt-Freeness" and, more generally, "Coercion Resistance" are

necessary properties in order to avoid the occurrence of voter coercion or of vote buying to violate the election. The existing systems use various forms of cryptographic and security protocols to ensure that these criteria are met in their implementation of an online election system.

Many of the existing systems begin by breaking the voting process up into multiple phases, and also partitioning the responsibilities for these steps among various different components. For example, the system developed by Cranor and Cytron, breaks the process up among four components: the "Registrar," "Pollster," "Validator," and "Tallier" [*2*]. In this scheme, the Registrar is "responsible for registering voters prior to an election or poll." The Pollster then acts during the actual election to send "human readable ballots to a voter, collecting the voter's responses," and perform "cryptographic function on the voter's behalf," among other features. The Validator verifies that a voter is registered and ensures "that only one vote is cast by each registered voter." Finally, the Tallier has the function of "collecting the voted ballots and tallying the results of the election or survey" [*2*]. The system implements various cryptographic protocols for the communication between these components and the voters' machines to ensure the security of the system and the satisfaction of the given criterion. This system is based closely on that proposed by Fujioka, Okamoto, and Ohta [*3*]. The system proposed by Karro and Wang [*1*] not only breaks the process up into different components, but also describes in detail different phases of the online voting scheme. This process uses six components, or "central facilities," in the voting scheme: "the *registrar*, the *authenticator*, the *distributor*, the *counter*, the *matcher*, and the *verifier*" [*1*]. In addition, the protocol consists of four different phases: the "Registration phase", "Pre-Voting phase", "Voting phase", and "Announcement phase" [*1*]. Again, this protocol uses various cryptographic procedures and encryption/decryption methods to ensure that the vote is administered fairly and no fixing or changing of votes can take place.

In achieving these criteria, most of the schemes utilize a set of cryptographic primitives to develop their election protocol and make it secure. These primitives include public-key cryptosystems, bit-commitment schemes, homomorphic encryptions, and mix-nets. These primitives allow the protocols to ensure the security of the information exchanged and the accuracy, privacy, and fairness that are necessary in a good election scheme. A more detailed description of these primitives and how they are used in some representative existing protocols is presented in our report entitled "A Survey of Electronic Voting Schemes," which we submitted as the result of our first semester work.

In this report, we have looked at a reflective sample set of protocols and explored their respective strengths and weaknesses. We have also outlined a list of the criteria that these schemes have suggested and that we see as being necessary for a safe and secure voting scheme. The final set of criteria that we determined was the most robust and complete includes:

- **Privacy**, which refers to the secrecy of each voter's vote being maintained throughout the election.
- **Eligibility**, which is realized if only eligible voters can vote, and they are only able to vote once.
- **Fairness**, which is provided if no party can gain information about any intermediate results during the election process.

- **Soundness**, also referred to as **robustness**, which is exhibited if faulty or dishonest behavior by any participants does not cause any unreasonable delay or invalidation of the whole election.
- **Verifiability**, which means that the announced results of the election can be checked. In a scheme that provides **individual verifiability**, voters can confirm that their own votes were received and counted, although they cannot confirm the overall results of the election. A scheme providing **universal verifiability** allows any observer to confirm that the results of an election are correct based on public transcripts generated during the voting process.
- **Receipt-freeness**, which is an important property required to prevent the practices of coercion, in which voters are threatened to vote in a certain way, and vote-buying, in which voters willingly accept some form of payment in exchange for their voting in a certain way. In an election scheme that is receipt-free, voters are unable to construct any receipt that can convince anybody of how they voted, so coercers or vote-buyers have no means of confirming that voters acted as they were instructed.
- **Coercion-resistance**, which extends upon receipt-freeness to prevent a number of additional coercion attacks.

We have attempted to replicate all of the positive and effective aspects of these sample protocols in the implementation that we have created. At the same time we have tried to improve upon any shortcomings that we found in their methods, in an attempt to create a more secure and efficient protocol for online voting.

Although there are many proposed schemes, we are not aware of many full implementations. Two that we have encountered are eVox [*6*] and Sensus [*7*], both based on the scheme by Fujioka, Okamoto, and Ohta [*3*]. As we have explained in our report on electronic voting schemes, this scheme is far from the most sophisticated available in today's current literature. Our goal of implementing a more feature rich protocol is novel, in this respect.

Technical Approach:

The final product of our work on this topic is an implementation of an existing online voting scheme that, while not fully functional for real world online elections, employs the various security features and attempts to achieve the desired criterion that we had outlined in our first semester report. In exploring the many existing schemes to determine the most robust and secure to use in our implementation, we found that the scheme proposed by Alessandro Acquisti [*5*] best satisfied these requirements. Acquisti's scheme comes closest of all that we surveyed to satisfying all of the desired characteristics that we laid out for a secure and robust voting scheme. In addition, it is the scheme that provides the easiest transition to implementation, as it makes the fewest physical assumptions and provides more efficient methods for achieving a secure scheme. For these reasons, we chose this scheme as the basis for our demonstration electronic voting system.

Our goal was to create an implementation of this scheme that, while not providing a full working system for running online elections, offers a framework that could potentially be used for implementing such a system. In addition, we hoped to gain further

insight into the challenges that arise in creating an online election system, and how they could be overcome.

Brief Overview of Voting System

Components

Our implementation is based on Acquisti's scheme, but also includes various changes and improvements to this original model. It is comprised of a set of Java applications that provide the functionality for the different components used in the Acqusiti scheme. There are three major components to this scheme, as can be seen in the Main Election Components diagram in the appendix. The first component is the voter application, which implements the functionality of the *Voter* entities described in the model. This application provides a user interface for the voter to interact with in participating in the election, as well as implementations of the various cryptographic protocols employed on the voter end of the scheme. The second major component is the administrator application. This implements the election *Authority* entities that Acquisti describes, and provides the functionality for running and overseeing an election. The administrator application provides the services of publicly distributing the election choices, distributing credentials to valid voters, and collecting and tallying the votes in the election. This application also implements the various cryptographic protocols that are used in making the election secure and in satisfying the fair election criteria. Finally, the *Bulletin Board* application provides the functionality of a publicly readable "broadcast channel with memory" [*5*] and allows for the fairness and legitimacy of the running of the election to be verified. It provides an interface into a database where all of the results and calculations of the election are stored and made available publicly for the purposes of verifying the election.

Phases of Election System

The voting system proceeds through three major phases, as can be seen in the Election Timeline diagram in the appendix. It is assumed that, before the election process begins, the administrators possess a list of eligible voters and their corresponding public encryption keys. This is assumed to be done in some sort of registration phase completed before our application is run. In the scheme proposed by Acquisti, the administrators are assumed to share, in a threshold manner, three sets of private encryption keys, the corresponding public keys being posted to the bulletin board and known publicly. The first two sets are El Gamal keys, implementing a homomorphic scheme, about which we will give more details later on, and the third set is a set of RSA keys. By a threshold scheme, it is meant that each of the administrators holds a share of each private key, such that a certain subset of the administrators must work together to decrypt any ciphertext, and any smaller subset cannot decrypt the ciphertext. After the sets of keys are generated, each of the administrators selects a random number and posts its encryption under each of the El Gamal keys to the bulletin board. This will be done once by each administrator for each of the possible choices in the election. Each of these encryptions will represent a share of an acceptable ballot, so that each voter will take one share corresponding to each administrator and combine them for the ballot that they wish to submit. Next, each

administrator selects another random number for each eligible voter, which will serve as a share of the credential for that voter. The administrator posts one encryption of this to the Bulletin Board, and sends another, under the other El Gamal key, to the corresponding eligible voter. Each administrator also presents proofs that they have used the same number for the two different encryptions for both the ballot shares and credential shares, using a proof such that they do not actually reveal any of the random numbers. By generating the acceptable ballots and the credentials in such a way, we ensure that no one administrator knows what values the ballots and credentials take, since they only generate a share of each. This ensures that a corrupt administrator would not be able to create and post fake votes using the credentials, since they would have to know the values for all of the credential shares in order to do so.

After the voter has received his credential share and proof from each administrator, he proceeds to the voting phase. In this phase, the voter combines the shares of the credential that he received from each administrator to get an encryption of the full credential. In addition, he takes from the bulletin board the shares of the ballot corresponding to the choice that he wishes to make in the election, and again combines these shares to get an encryption of the ballot under the El Gamal key. He combines the encryption of the ballot with the encryption of the credential, encrypts it again under the threshold RSA key, and posts it to the bulletin board. After all voters have voted or a certain time is reached, the voting phase ends.

The final phase is the tallying phase, in which the administrators jointly decrypt submitted votes and tally the totals. To do this, they first decrypt the outer RSA layer of encryption for each vote. They then pass this list of votes through what is known as a mix net, which permutes the votes in such a way the no one can connect the vote to the voter who submitted it. In addition, the administrators take the shares of the credentials that were posted to the bulletin board in the first phase and combine them to get a list of encryptions of valid credentials, which they pass through another mix net. Finally, they take this list of valid credentials and check it against the list of votes submitted to find which votes had valid credentials. The increase the tally for the corresponding ballot choice each time they find a vote with a valid credential. Once they have counted every vote, they post the results of the election and the election is completed.

Implementing the Required Components

Threshold El Gamal Cryptosystem

The portion of the Acquisti scheme in which we most departed from the protocol proposed in his paper was in the use of the El Gamal cryptosystem for the encryption of the ballot and credential shares produced by the administrators. A brief description of this system is given in Appendix A. In the original scheme, the ballot and credential shares are encrypted under a different threshold encryption system, called the Paillier cryptosystem. Both El Gamal and Paillier share certain features, in that they can be employed in a threshold decryption manner and also can be made to be homomorphic, in that the multiplication of ciphertexts would correspond to the addition of the underlying plaintexts. In fact, Acquisti mentions in an appendix to his paper that his scheme could in fact be implemented with El Gamal instead of Paillier. He cites the better efficiency of the Paillier system as his reason for using it. We chose El Gamal, however, for a couple

of reasons. First, we were more familiar with El Gamal and felt that it would be simpler to implement and to make efficient. In addition, we believed that, for our purposes of studying a demonstration voting system, the differences in efficiency between El Gamal and Paillier were not appreciable. Also, we implemented the encryptions of the votes so that it was not necessary to take a discrete log in the decryption phase, as it had been in the original scheme. When it is necessary to take such a discrete logarithm, the Paillier scheme offers significantly improved efficiency over the El Gamal scheme. In light of this, the El Gamal system provided a sufficiently efficient and easy to implement scheme, relative to Paillier.

In addition to implementing the system using El Gamal instead of Paillier, we also added a new feature, not mentioned in the Acquisti proposal, to the Threshold El Gamal system that was intended to improve the security of the system. This feature addressed the generation of the El Gamal parameters and shared secret keys, which in many applications is assumed to be done by a trusted third party dealer. However, for our scheme we found a protocol, proposed by Pedersen, [*9*] that allows the shared secret key to be generated in a distributed manner, so that no party ever has possession of the entire secret key, but each administrator gets a share of the key that can be used to decrypt in a threshold manner. This improves the security of the scheme, as it is no longer necessary to assume that there is a trusted third party that generates the keys, and thus the security of the El Gamal is fully guaranteed as long as a threshold number of administrators are honest, even if all other administrators act maliciously.

In the scheme proposed, the shares of the ballots and the shares of the credentials produced by the administrators are each encrypted twice, under two different El Gamal keys. In the case of the ballot shares, both encryptions are posted to the bulletin board, and in the case of the credentials, one encryption is posted and the other secretly sent to the voter. So, in order for the scheme to be provably legitimate, it is necessary for each administrator to prove, without revealing any information, that the two encryptions of the ballot shares under different keys contain the same underlying plaintext, and similarly for the credential shares. In addition, for the proof involving the credentials share, the proof must be done in a designated verifier manner, so that only the voter for whom the credential is intended can verify that the proof is valid. This prevents the voter from being coerced by an outside force to reveal his credentials, as the voter can provide a false credential and false proof of its validity to the coercer. Acquisti provides a way for the administrator to do this under the Paillier scheme, but does not provide any way of proving this using the El Gamal scheme. Thus, we had to develop our own method of providing this proof, based on similar proofs presented by Hirt and Sako [*10*] and in the Cryptovote paper [*11*]. A more detailed description of the proof of equivalent plaintext and the designated verifier proof are given in Appendix B.

Threshold RSA Cryptosystem

Threshold RSA encryption [*15*] is used by voters when submitting their votes as a means of making the submitted vote secure against certain attacks. The purpose of this extra layer of encryption is to prevent an attack in which an adversary intercepts the submitted vote from the voter and alters it in such a way that the credential remains the same but the ballot has been changed, thus changing the voters intended vote. This

would be possible if the vote were just encrypted under the El Gamal keys, but is prevented by the extra layer of RSA encryption.

In our implementation, we made the choice, unlike in the case of the threshold El Gamal system, of not generating the RSA key in a distributed manner. Acquisti's scheme does not address the problem of the generation of these RSA keys, so this was one of the choices we had in our implementation. Thus, we assume there is a trusted third party dealer that creates the keys and distributes a share of the private key to each administrator. The major reason that we chose not to implement this distributed key generation for RSA [*12*] was that it was far more complicated and more inefficient than the distributed key generation procedure for El Gamal. In a real implementation of a system, this distributed key generation would provide additional security, as it removes the trusted third party and allows for a system where no single party possess the full private key. For the purposes of our demonstration applications, however, we chose not to implement this due to the cost in efficiency.

Another feature that we added to our implementation but was not present in the original Acquisti scheme deals with the administrators proving that they have correctly decrypted their share of a ciphertext. In the threshold system, each administrator uses its share of the private key to produce a share of the decryption of the ciphertext. A certain subset of these shares must be valid and are then combined to produce the full decryption. Each administrator must thus provide a proof that he decrypted his share correctly, for each ciphertext decrypted. We implemented a batch proof system, developed by Aditya, et al, [*13*] that allows each administrator to prove in a single proof that he decrypted a whole set of ciphertexts correctly. Thus, when each administrator has to decrypt a share of each submitted vote, he can present a single proof that shows he did this correctly for all votes. This greatly improves the efficiency of the voting system, especially in a real world election in which potentially millions of votes could be submitted. This batch proof method is used with both the RSA and El Gamal decryptions in our implementation.

Mix Net Servers

The protocol of a mix net server provides a way to achieve privacy and anonymity of votes by taking the list of submitted votes and permuting and re-encrypting it in such a way that the connection between the voter who submitted the vote and the final vote that is tallied is hidden. This is achieved by having each of the different administrators generate a secret permutation on the list of votes, then permuting and re-encrypting the votes in sequence with the other administrators. Specifically, the first administrator applies its secret permutation to the original list of votes and re-encrypts them by changing the encryption randomness, then passes this to the second administrator, which performs the same task and passes the output to the third, and so on. By the end of the process, no one can trace a vote in the final list to the original vote in the first list.

One of the facets of a mix net that our implementation addresses but was not touched on in the Acquisti paper has to do with proving the validity of mixing. In order for the mix output to be trusted, each administrator must prove that it mixed and re-encrypted in a legitimate way, so that the underlying plaintexts of the final list of messages are indeed simply a permutation of the plaintexts in the first list. However, they cannot simply reveal the permutations they used, as that would violate the privacy of the

process. Thus, we searched through various different proposed methods for proving correct operation of a mix net. We decided on a protocol, developed by Jakobsson, Juels, and Rivest [*14*] that was both efficient and probabilistically secure for this purpose. Essentially, the protocol works by having each administrator randomly reveal a certain fraction of their permutation input-output correspondences, so that we still have the property that no vote can be traced from the first list to the final, but at the same time we are assured with large probability, due to the fact that the correspondences that need to be revealed are random, that each server mixed the list legally. As opposed to some of the other existing schemes for providing this proof, which need to provide a proof for each administrator and for each output element in the list, the scheme we chose allows the proof to be given for all elements in the output list of one administrator at once. This allowed us to prove in a very efficient manner the correct operation of the mix net, even when the list to be mixed is very long, as in the case of the list of votes. This is the major contribution our implementation makes to this portion of Acquisti's scheme.

Technologies and Tools Employed in our Implementation

Our implementation was constructed using Java, which provided us with many conveniences. Perhaps the most important was its existing support for arithmetic operations involving unboundedly large integers, which is provided by its BigInteger class. This class' well-implemented methods for modular exponentiation, multiplication, and modular inversion were used heavily, and it integrated well with our methods for computing hashes and writing the numbers to a database, thanks to its support for conversion to and from byte arrays.

Java's Cryptography Architecture API provides a simple interface for performing other common tasks. The API is designed to allow various algorithms for producing cryptographic hashes, random numbers, signatures, and encryptions to be enabled under one common interface by plugging in an appropriate "provider" of the underlying algorithms' implementations. The Cryptix Project [*16*] makes available one such provider that implements a number of algorithms including RSA encryption/decryption with secure padding, RSA signatures, and various hash algorithms including SHA-1 and its larger variants, as well as algorithms for public/private key generation. These were used extensively in our implementation: SHA-512 hashes implemented in Cryptix were used in commitment schemes, as well as in generation of challenges for non-interactive proofs; the non-threshold RSA encryption and signature systems called for in the election scheme were all implemented using the Cryptix provided algorithms, which perform well and conform to specifications outlined by RSA Security [*17*]**.**

Communication between the various participants in the election scheme is done over TCP/IP, which is easily accomplished with Java's support for sockets and streams. Its support for threads and synchronized methods is used by both the bulletin board and administrator modules to support multiple clients at once, and to manage the reading and writing of information as it is posted to the bulletin board while avoiding race conditions and other synchronization issues.

Originally, we had thought of implementing the voter client using a web-based interface, which would have provided convenient access to voters. However, in the end we chose to make the voter client a Java application, like the other modules. This simplified the design, and had a few other advantages. For instance, to provide RSA

encryption of the posted votes in a browser-based interface, either the browser would have to encrypt the data itself, or the host of the web application would have to be trusted to do so while the plaintext was sent over HTTP using a secure socket layer, both much more complicated solutions. Also, a case can be made for using stand-alone client software rather than a commonly accessed web interface based on security, since a web interface opens the possibility of spoofing attacks in which a false voting site could be presented to voters, or a corrupt administrator himself could present a dysfunctional site. While trust of a voter's own client software is still an issue using a stand-alone application, the voter is free to obtain the software from a trusted source, and savvy voters could even possibly program their own voting clients to conform to the protocol.

We did suffer one disadvantage in our use of Java, specifically its Cryptography Architecture API: the default installation of the Java VM places limits on the strength of cryptography that can be used when running applications and the key sizes we were using exceeded them, requiring us to install Sun's policy extender on our machines in order to lift these limits. This made testing our software on multiple systems difficult, since we were not aware of many other machines on which we would have the authority to modify the Java runtime environment in order for us to be able to test our code.

The other main tool in our implementation is a MySQL database server which provides permanent storage of the election data. Our implementation would probably have been demonstrated just as well without it, since the election data could have been stored in memory by the bulletin board application, but any fully developed election system would certainly provide a permanent record of the transmissions so that the election results could be verified and examined by others afterwards. Our integration of a database server is a step towards this. We would like to provide a separate module that could query the database and independently verify the election data that exists on it, however we are not sure at the time of writing whether this will be ready before the demonstration.

Other Challenges and Design Decisions

A number of design issues that are not discussed in detail by Acquisti were encountered during the planning of our implementation. His election scheme, as well as many others, begins by assuming a distribution of various keys to the election participants. In addition to the three sets of threshold keys which are shared among the administrators, there are El Gamal and RSA key pairs associated to each registered voter, as well as two sets of RSA key pairs for each administrator, one of which is used for encryption, the other for providing digital signatures of the administrator's posted data. The overall security of the election scheme relies on the secure and trusted distribution of all of these keys. Rather than attempt to accomplish this within the system we implemented, we started off making the same assumption that participants already had their own private keys as well as the public keys of other participants. In practice, this is a reasonable assumption. Our system achieves it by using utilities which generate the required keys for participants and write them to files, so that the files containing the public components can be distributed beforehand by means that meet the security requirements of the election and that the election participants would trust.

The stage at which voters communicate with administrators to receive their encrypted credentials presented other issues that were not fully discussed in Acquisti's scheme. A method by which voters could authenticate themselves to the administrators had to be implemented. We chose to have voters authenticate themselves by providing a zero-knowledge proof that they know the private key corresponding to their public El Gamal key. This establishes the voter's identity (assuming their private key remains private) and also helps ensure the effectiveness of the designated verifier proof that the administrator sends, since it confirms that the voter has the ability to construct fake proofs of what credentials were received from the administrator. Additionally, each voter has a unique RSA key pair which the administrators use to encrypt the credential shares and designated verifier proofs that are sent to the voter.

One weakness of Acquisti's scheme which we perceived was that during the voter-administrator interaction, a voter could be denied a correct credential share from a corrupt administrator. The voter would immediately realize this (the designated verifier proof would not verify, or the administrator simply would refuse to communicate at all), however there would be little recourse since there is no way for the voter to prove the authorship of the designated verifier proof. We explored whether we could modify the election scheme so that only a certain threshold of credentials would have to be collected and combined by a voter in order to form a valid vote, possibly using secret sharing techniques similar to those in the El Gamal threshold key generation, but were unable to find a way. This is certainly an area of the Acquisti scheme that could be explored and improved.

The implementation of the bulletin board described in the scheme also brought with it some design decisions and assumptions. The bulletin board is supposed to provide a repository of all the current election data which should be readable by anybody, but the existing contents should not be altered or deleted. Achieving these requirements is very difficult without making some additional assumptions. For instance, if the bulletin board is implemented by designating a server which receives all the data and records it as an official record (as in our system), an untrustworthy bulletin board server could perform a number of attacks, including withholding information from certain voters, or denying certain voters the ability to post their votes (although the information upon which it could selectively discriminate is limited to packet headers, since the data of the posted vote contains no identifying information). A fully developed system might minimize the amount of trust required by distributing the bulletin board across multiple servers in some way, so that only a fraction would need to act honestly. Implementing such a system would be a very large task, so we have made the simplifying assumption that our bulletin board server does not perform the mentioned attacks.

We also trust the bulletin board to perform other tasks in our implementation. One of them is the combining of decryption shares that are posted by the administrators under the various threshold cryptosystems. This could be performed independently by any observer with access to the public bulletin board contents, however for purposes of automating the election process it was assigned to the bulletin board. Since the results can be independently obtained, the bulletin board cannot falsify them without being caught.

Our implementation also trusts the bulletin board to coordinate the timing of commitment protocols between the administrators in the key generation and mix net phases. It is responsible for monitoring which commitments have been posted to the

bulletin board and then notifying the administrators when all the commitments have been made public so that they may then be revealed. This was another task assigned to the bulletin board as a matter of convenience, since the administrators are capable of monitoring the published commitments themselves.

Conclusion:

Our final application provides a demonstration implementation of the Acquisti scheme that satisfies the majority of the criteria that were required for a secure electronic voting scheme. There is still a great amount of work that would have to be done to provide a fully developed system for conducting online elections, such as implementing a more robust bulletin board system, implementing a distributed RSA key generation protocol, exploring different authentication methods, and attempting to resolve some of the remaining weaknesses of the scheme which we have mentioned. Still, our implementation accomplishes a large part of what is involved in such a system, and has given us a clear idea of what the main remaining challenges are to providing secure online elections.

References:

1. Karro J.; Wang J.; Towards a Practical, Secure, and Very Large Scale Online Election. *Computer Security Applications Conference, Proceedings,* 1999, *15th Annual*, 161-169.

2. Cranor L. F.; Cytron R. K.; Sensus: A Security-Conscious Electronic Polling System for the Internet. *Proceedings of the Hawaii International Conference on System Sciences* 1997, *Volume 3*, 561-570.

3. Fujioka A.; Okamoto T.; Ohta K.; A Practical Secret Voting Scheme for Large-Scale Elections. Proceedings of *Advances in Cryptology, Lecture Notes in Computer Science*, 1993, *Vol. 718*, 244-251.

4. Juels A.; Jakobsson M.; Coercion-Resistant Electronic Elections, 2002. http://citeseer.nj.nec.com/555869.html.

5. Acquisti A.; Receipt-Free Homomorphic Elections and Write-in Voter Verified Ballots, Technical Report 2004/105, International Association for Cryptologic Research, May 2004, http://www.heinz.cmu.edu/~acquisti/papers/acquisti-electronic_voting.pdf.

6. Herschberg M.A.; Secure Electronic Voting Over the World Wide Web, 1997, http://theory.lcs.mit.edu/~cis/theses/herschberg/abstract.pdf

7. Cranor L. F.; Cytron R. K.; Sensus: A Security-Conscious Electronic Polling System for the Internet. *Proceedings of the Hawaii International Conference on System Sciences* 1997, Volume 3, 561-570.

8. ElGamal T.; A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *Advances in Cryptology – CRYPTO '84*, Springer-Verlag , 1984, *Vol. 196* of *LNCS*, 10 – 18.

9. Pedersen T.; A Threshold Cryptosystem without a Trusted Party. Advances in Cryptology - Eurocrypt '91, Springer-Verlag, 1991, Vol. 547 of LNCS, 522-526.

10. Hirt M.; Sako K.; Efficient Receipt-Free Voting Based on Homomorphic Encryption. Lecture Notes in Computer Science, 2000, Vol. 1807, 539 - 556 .

11. Smith W.; Cryptography Meets Voting, 2004. http://www.math.temple.edu/~wds/homepage/cryptovot.pdf

12. Frankel Y.; MacKenzie P. D.; Yung M.; Robust Efficient Distributed RSA-Key Generation. Proceedings of STOC 98, 1998.

13. Aditya R.; Peng K.; Boyd C.; Dawson C.; Lee B.; Batch Verification for Equality ofDiscrete Logarithms and Threshold Decryptions. Applied Cryptography and Network Security – ACNS '04, 2004, Vol. 3089 of LNCS, 494-508.

14. Jakobsson M.; Juels A.; Rivest R.; Making mix nets robust for electronic voting by randomized partial checking. In Proc. of USENIX Security'02, 2002, 339-353.

15. Fouque P; Poupard G; Stern J.; Sharing decryption in the context of voting or lotteries. Financial Cryptography '00, Lecture Notes in Computer Science, 2000

16. http://www.cryptix.org

17. RSA Security Inc. Public-Key Cryptography Standards (PKCS), 2002.

Appendix A: El Gamal Cryptosystem

One example of a public-key cryptosystem utilized in many of the election protocols, and also used as a homomorphic encryption scheme, is the El Gamal encryption scheme, presented in [*8*] In this system, the keys are based on a large prime p and a generator g of Z_p^*. The private key is a value x, chosen from Z_{p-1}, and the public key is $y = g^x$. To encrypt a message m, the pair $(a, b) = (g^r, m\, y^r)$, where r is a random value chosen from Z_p, called the randomness, is calculated by anyone who knows the public key. To decrypt, the holder of the secret key calculates $\frac{b}{a^x}$, which will retrieve the message m. Without the secret key, it is believed to be very hard to decrypt the ciphertext, as it would be necessary to find the exponent x from the value $y = g^x$. This is a problem known as the discrete log problem, and there is no known way to solve this problem in polynomial time. To make the scheme useable with homomorphic encryption, the following protocol is applied. To encrypt a message m, one instead

encrypts G^m, where G is another generator of Z_p^*. If encryption is used this way, then the component-wise multiplication of ciphertexts corresponds to the addition of plaintexts. Given $(a_1, b_1) = E(m_1)$ and $(a_2, b_2) = E(m_2)$, it can be seen that $(a_1 a_2, b_1 b_2) = E(m_1 + m_2)$ in the homomorphic variant. This cryptosystem also supports a function called *threshold decryption*[*15*]. Similar to secret sharing, it provides the functionality that a group of parties each get a share of the secret key, and only when $t + 1$ or more of the parties, where t is some security parameter, put their shares together can they decrypt a message. In this case, the secret key itself is shared in a secret sharing scheme, where each party gets a share x_i of the key. Then, each member can decrypt using their share, computing $\frac{b}{a^{x_i}}$, and any group of $t + 1$ or more parties can combine these values to find the plaintext message, while any group of t or fewer parties can learn nothing about the plaintext.

Appendix B: Proofs of Equivalent Plaintexts

Here, we describe how a prover can prove to a verifier that two encryptions it has produced, under two different keys and using two different encryption randomness values, have the same underlying plaintext. This is used in the pre-voting phase when the administrator proves that the two different encryptions of the ballot shares contain the same plaintext m, and similarly for the credentials shares. Let $(\alpha_v, \beta_v) = (g^{r_1}, h_v^{r_1} m)$ and $(\alpha_c, \beta_c) = (g^{r_2}, h_c^{r_2} m)$ be the two encryptions, known to both prover and verifier. First, the prover randomly selects a value r and secretly computes $(\alpha_1, \beta_1) = (g^r, h_v^r m)$ and $(\alpha_2, \beta_2) = (g^r, h_c^r m)$. The prover then sends to the verifier the following six values:

$$a_1 = \frac{\alpha_v}{\alpha_1}, \quad a_2 = \alpha_1 = \alpha_2, \quad a_3 = \frac{\alpha_2}{\alpha_c},$$
$$b_1 = \frac{\beta_v}{\beta_1}, \quad b_2 = \frac{\beta_1}{\beta_2}, \quad b_3 = \frac{\beta_2}{\beta_c}$$

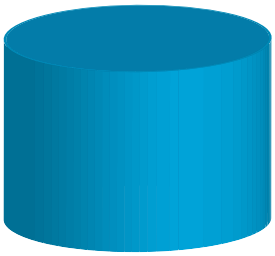
Upon receiving these values from the prover, the verifier checks that $a_1 \cdot a_2 = \frac{\alpha_v}{\alpha_c}$ and that $b_1 \cdot b_2 \cdot b_3 = \frac{\beta_v}{\beta_c}$. Then, the verifier selects a random value c from the set {0, 1, 2} and sends c to the prover. Based upon this value c, the prover produces a proof of equivalence of discrete logarithm for two values. If $c = 0$, the prover presents a proof that $\log_g a_1 = \log_{h_v} b_1$, which shows that (α_v, β_v) and (α_1, β_1) contain the same plaintext. If $c = 1$, the prover presents a proof that $\log_g a_2 = \log_{\frac{h_v}{h_c}} b_2$, which proves that (α_1, β_1) and (α_2, β_2) contain the same plaintext. Finally, if $c = 2$, the prover provides a proof that $\log_g a_3 = \log_{h_c} b_3$, which shows that (α_2, β_2) and (α_c, β_c) contain the same plaintext. When the proof is used for ballot shares, each of these second proofs is a simple proof of discrete log equivalence, which is described in detail in [*11*]. When done

for credentials shares, each of the second proofs is done in a designated verifier manner, which will be described below. Thus, in this process, the prover shows to the verifier that one of three plaintext equivalences is true. We see that if all three are true, then we will have that (α_v, β_v) and (α_c, β_c) contain the same plaintext, which is the desired result. So, since the challenge *c* is picked randomly, a prover who is lying about one of the equivalences can fool the verifier with a probability of $\frac{2}{3}$. If we repeat the above procedure λ times, we see that a lying prover only succeeds with a probability of $(\frac{2}{3})^{\lambda}$, which is a probability that shrinks quickly if we repeat enough times. Thus, the verifier can be sure with a large probability that the plaintext equivalence is true after a number of repetitions of this proof. Also, we see that this is a zero knowledge proof, since a simulator who does not know the underlying plaintexts or randomness values can produce a transcript of the proof that is indistinguishable in its distribution from a real transcript. This can be done since the simulator can guess the value that *c* will take, and adjust the $a_1, a_2, a_3, b_1, b_2,$ and b_3 values so that the appropriate proof is true given that *c*. The other two proofs of plaintext equivalence will not be true in the simulator, but since it only needs to show the proof selected by the value of *c*, the simulator passes the proof. Thus, we see that the proof does not reveal any information about the plaintexts, other than the desired equivalence, since a simulator that knows nothing more about the plaintexts than their equivalence can give an accepted proof.

The following designated verifier proof for discrete log equivalence is based on the normal proof of equivalence presented in [*11*] and the designated verifier proof shown in [*10*]. In the following procedure, the prover wishes to show to a verifier, who has a known public key h_v and a secret key s_v, where $h_v = g^{s_v}$, that $\log_g x = \log_h y = l$, for some value *l*. The first step in the proof involves the prover randomly selecting three values, *d*, *r*, and *w*. The prover then computes $a = g^d, b = h^d,$ and $s = g^w h_v^r$ and sends *a, b*, and *s* to the verifier. The verifier then picks a random challenge value *c* and sends this to the prover. The prover then computes $z = d + l \cdot (c + w)$ and sends *z*, *r*, and *w* to the verifier. The verifier checks that $s = g^w h_v^r$, $g^z = a\, x^{c+w},$ and $h^z = b\, y^{c+w}$. The second and third checks show the indeed the discrete logs are equal and are known by the prover. The first check provides the designated verifier aspect of the proof. Since the verifier knows the secret key s_v, it can be shown that he can select different values for *w* and *r* after the random challenge has been received so that the three checks will work even if the discrete logs are not equal. Thus, if a coercer were to force the verifier to repeat the proof to the coercer, the verifier could produce a false transcript with a different value for *x* and *y* that would still pass the checks. Thus, the coercer could not be sure that the proof presented by the designated verifier is valid. This is useful in the context of our election scheme in that it allows the voters, who are the designated verifiers in our case, to provide fake credentials and a fake proof to a coercer that is attempting to steal their credentials and vote for them. So, a voter could fool the coercer in to thinking that he gave the coercer correct credentials, and then vote for his desired choice anyway with his real credentials.

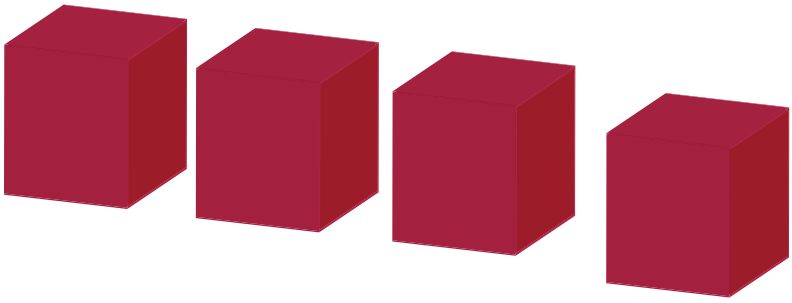
Main Election Components

Bulletin Board

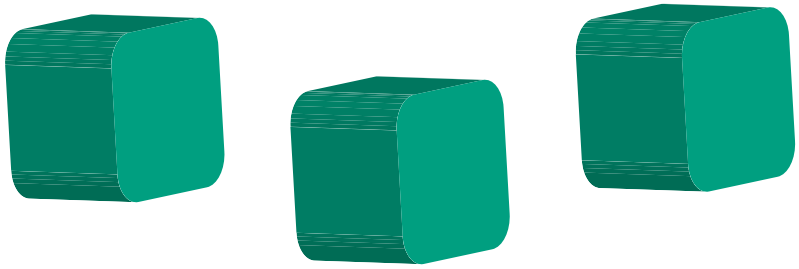


- Publicly writeable/readable database. Non-erasable.
- Public information, proceedings and results of election are posted and can be seen and verified by any observer.
- Distributes election parameters to authorities and voters.
- Coordinates the administrators and steps in the election

Administrators



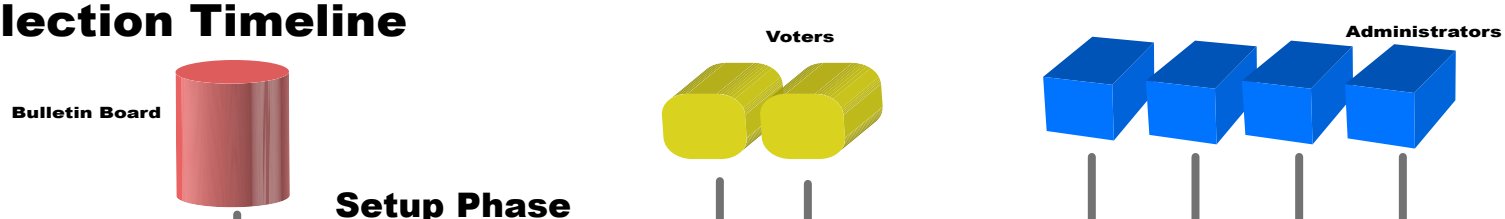
- Set of authorities that oversees the running of the election
- Shares three sets of decryption keys, two ElGamal keys, one RSA key
- Creates ballot and credential shares used in election.
- Distributes credential shares to voters, encrypted under first ElGamal key, post second set to BB, encrypted under second ElGamal key, and posts two sets of ballot shares to BB encrypted under each ElGamal key
- Performs Mix Net operation on credential shares posted to BB.
- Performs Mix Net operation on list of votes submitted.
- Decrypts RSA layers from votes submitted
- Searches through submitted votes, compares to valid credentials and ballots, tallies election.



Voters

- Components that represent the participants/voters in the election.
- Receives credentials from Administrators, along with proofs of equivalence with set posted to BB.
- Selects desired ballot choice in election, downloads corresponding ballot shares from BB.
- Combines ballot shares from BB, which are encrypted under first ElGamal key, with credentials received from Administrators, also under first key.
- Encrypts resulting ciphertext under RSA key, which represents vote.
- Posts vote to BB.

Election Timeline



Setup Phase

El Gamal Key Generation for both Votes and Credential Key Pairs

Administrators generate random polynomials to be used in secret sharing

Administrators publically commit to their polynomials via cryptographic hashes and modular exponentiation in

Secret shares are privately distributed between the administrators. The shares are verified against the published commitments

Verification keys are posted to the bulletin board

Threshold RSA decryption and verification keys are distributed to the administrators

El Gamal encrypted ballot and credentials shares are posted, along with proofs of plaintext equivalence

Voting Phase

Voters retrieve election parameters from the bulletin board

Voters assemble their own El Gamal encrypted credential from shares obtained from each administrator

Encrypted ballot shares are obtained from the bulletin board. Voters select the shares corresponding to their voting choice and combine them with their credential

Voters encrypt their vote under the threshold RSA public key and post it to the bulletin board

Tallying Phase

Administrators sign the bulletin board to certify its contents

Credentials Mix Net

Administrators pass the published combined credentials through a mix net

RSA Decryption of Votes

Administrators publish RSA decryption shares of posted votes, along with a batch proof of correct decryption

Bulletin board assembles a set of proven decryptions and combines them, retrieivng the underlying El Gamal encrypted votes

Votes Mix Net

Mix net is repeated with votes

El Gamal Decryption of Votes

Administrators publish El Gamal decryption shares of votes, along with a batch proof of correct decryption

Matching

Credential and ballot encryptions are systematically combined, decrypted, and matched against the vote plaintexts; any matched credentials are removed, and the matched vote is tallied