A Cryptographically Secure Voting System for the MIT Community

Fernando Trujano
trujano@mit.edu

I. Introduction

Every year, student groups and organizations throughout MIT hold elections in order to choose their leaders and representatives. Voting allows each individual to voice their opinion and feel like they are having an impact in the organization. Many times, votes are cast anonymously to protect the identity of the voter and ensure that other factors such as peer pressure do not influence a voter's decision. Furthermore, to guarantee the integrity of the system it is important that these elections are run as fairly as possible, without one party having total control over the results.

An ideal system would allow each individual to vote anonymously and verify that their vote was indeed counted and that all of the votes were counted correctly. Unfortunately, systems currently used at MIT fail to provide any guarantee of fairness or anonymity that an ideal system would call for.

I surveyed existing systems at MIT and found that many dorms still rely on paper ballots, which are submitted by residents and counted by a single party. This system does not guarantee anonymity (since one could for example, identify a ballot based on the pen used or the handwriting) or fairness (since whoever is counting the votes could purposely report incorrect results).

Some current systems, such as the one used by the Inter Fraternity Council (IFC), allow for individual verification of election results, but give up voter anonymity as a result. In the IFC, eligible voters meet to discuss an issue and then publicly vote by raising their hand. While this method may be okay for minor votes, more difficult decisions should be made anonymously for reasons explained above.

Other systems, such as the electronic one used in Simmons Hall, correctly guarantee anonymity, but fail to provide a way for the public to verify the results of the election and instead rely on a single party to tally and report the results. A corrupt party could intentionally or accidentally alter the results of an election.

As seen above, there are many desirable properties for an electronic voting system, which can be summarized by the following: privacy, unreusability, eligibility and verifiability. **Privacy** in this context is synonymous with anonymity. Voters should be able to cast their votes based on their beliefs without allowing others to see their vote. To maintain the integrity of the voting system, voters should be allowed to vote only once and their votes should be **unreusable**. For similar reasons, a desirable voting system should ensure that only the correct **eligible** users are allowed to submit their vote. Finally, voters will trust a system if they can individually **verify** that their vote was counted correctly and that the overall election results were computed fairly.

In this paper, I implement a cryptographically secure voting system for the MIT community. The system is cryptographically secure in that it uses a combination of cryptographic protocols to guarantee privacy and verifiability, making it computationally infeasable for an attacker to influence the results of an election. The design for the system is based on a paper by Fujioka and Okamoto [1] with simplifications and changes to better suit the MIT community. The main goal of the system is security: voters should be confident that their votes counted and can verify/audit the results of an election. With this system, the MIT community will be able to provide a consistent, secure, and anonymous voting experience for any type of election.

II. Design

This secret voting scheme achieves voter privacy while allowing verifiability by all parties, even in the case of malicious organizers and administrators. The system consists of a two-part voting stage and three parties, the voter, the administrator and the counter. An overview of the system is shown in Figure 1.

Voters communicate with the administrator and the counter in order to cast a secure ballot. This scheme relies on blind signatures, which are explained in detail in [5]. A voter first decides on his/her vote and writes it down on a ballot. A voter then blinds the ballot using a sufficient large randomly generated integer and sends their blinded ballot, as well as a form of authentication/ID to the administrator.

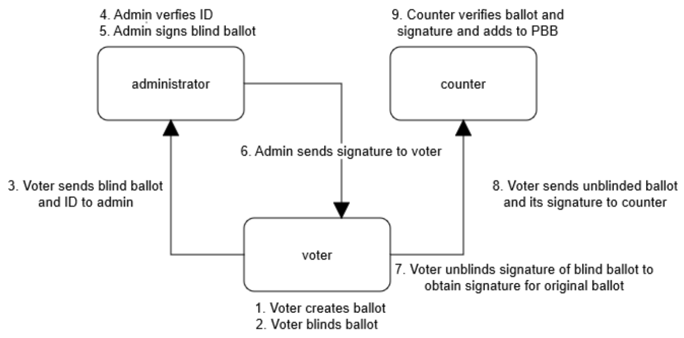

4. Admin verfies ID
5. Admin signs blind ballot
9. Counter verifies ballot and signature and adds to PBB
administrator
counter
6. Admin sends signature to voter
3. Voter sends blind ballot and ID to admin
8. Voter sends unblinded ballot and its signature to counter
voter
7. Voter unblinds signature of blind ballot to obtain signature for original ballot
1. Voter creates ballot
2. Voter blinds ballot

Fig. 1. Overview of the voting process. Parties are represented by rectangles while communication/messages are shown by arrows

The administrator is responsible for authenticating voters and signing their ballots. When an administrator receives a blinded ballot from a voter, the admin checks the voter's ID and their eligibility to vote and signs the blinded ballot if the voter is eligible. The administrator then sends the signed ballot back to the voter. Upon request, the administrator can give anyone a list of all blinded ballots that he/she has signed for this election. At this point, the voter can unblind the signed message in order to obtain a valid signature for their original ballot. The voter then sends their original ballot, along with the matching signature from the administrator obtained above through an anonymous channel to the counter.

The counter is in charge of receiving ballots, verifying them and counting the results. Upon receiving a ballot and signature from a voter, the counter verifies the ballot using the administrator's public key. If both the ballot and the signature are valid, the counter adds the ballot and the signature to a publicly accessible board (the public bulletin board or PBB). At the end of the election anyone can go and check the bulletin board to make sure that their vote was counted as well as the general result of the election.

A. Design Security

Many of the security proofs follow from that of the original design, but I will briefly mention them again here.

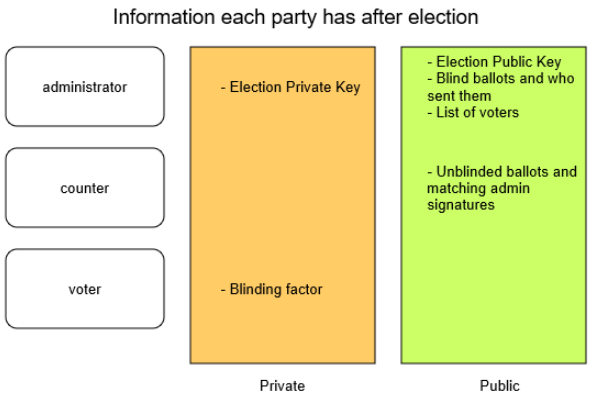
Eligibility - The administrator checks that a user is allowed to vote by authenticating them with their ID. Therefore, assuming the security of the ID mechanism, a person who is not allowed to vote will not be authenticated by the administrator.

Unreusability - Before signing a blinded ballot, the administrator checks that the voter is eligible to vote in the election. Once a voter has received a signature from the administrator, the administrator will not sign another ballot from that user. This way, voters can vote only once. A malicious or compromised administrator may purposely sign more ballots for a specific user, allowing them to vote more than once. In this case however, there will be a mismatch between the number of voters in the election, and the number of ballots cast allowing anyone to detect an inconsistency in the election and invalidate the results.

Privacy - In the first part of the voting, the voter sends a blinded ballot to the administrator, along with identifying/authentication information. Since the ballot is blinded, there is no way for the administrator to know the contents of the ballot, even if he/she compares their signature of the blinded ballot to the signatures of the unblinded ballots made available by the counter.

It is important to note that all communication between voters and the counter happens though an anonymous channel and thus the counter is never able to know who is sending the ballots. Therefore, there is no way to prove that any one ballot belongs to a specific individual.

Verifiability - Since all ballots are made public , anyone can visit the public bulletin board in order to make sure their vote was counted and verify the results of the election. Figure 2 shows what information is available to each party after the election.



Information each party has after election
administrator
counter
voter
- Election Private Key
- Blinding factor
- Election Public Key
- Blind ballots and who sent them
- List of voters
- Unblinded ballots and matching admin signatures
Private
Public

Fig. 2. Information that each party has after the election and whether that information is public or not

III. Implementation

This system is implemented as a single web app running on a node.js server. The backend uses mongodb to store election information such as voter data and ballots. Currently, admin keys are stored in the server's filesystem. As this system is designed for the MIT community, MIT certificates are used for authentication throughout the system when needed. The current flow of an election is as follows:

1) An organizer (i.e., any MIT community member) authenticates to the election server via MIT certificates and creates an election by listing the kerberos's of voters.
2) Voters create their ballot, which may include write-ins, and blind it using the provided open source python script (see Appendix A), or though their own means if preferred.
3) Voters receive an election ID from the organizer and use this to communicate with the right administrator in order to get their blind ballot signed.
4) Once the voters authenticate with the administrator, and the administrator verifies their eligibility to vote in the election (as specified by the organizer upon the creation of the election) the administrator signs the blind ballot and sends it back to the voter. The voter unblinds the ballot using the unblind python script (see Appendix B) or through their own means and obtains a valid signature for the unblinded ballot.
5) The voter sends the unblinded ballot and signature over to the correct counter using the electionID from step 3. Note that the counter need not know who the voter is at this point, so for privacy the voter is welcome to do this step on any device, or through a more anonymous channel, such as the Tor network [2].
6) The counter verifies the signature and adds the ballot to the public bulletin board.
7) Once everyone has voted, the organizer can decide to end the election, which reveals the ballots and their signatures to anyone that requests it for verifiability of the results.

In order to achieve a strong cryptographically secure voting system, many design choices had to be made. For example, the system trades off usability for strong security. Voting now requires four steps (blind, send, unblind, send) instead of one, but provides guaranteed privacy and verifiability as a result. The system purposely separates these steps to further increase user confidence in the system since all blinding and unblinding operations are performed outside of the browser.

Since the system is tailored for the MIT community, it makes use of MIT certificates for authentication, as these are a proven way to authenticate users. I used a modified version of the open source mit-cert-auth package [7] to authenticate MIT certificates from a non-MIT server. This package uses a shared secret between the non-MIT server and a small PHP script on an athena account. Furthermore, since the system allows anyone to create an election, a new administrator key pair is created for each election to prevent voters from anonymously casting ballots from other elections in order to influence the results.

IV. Future Work

As it stands currently, there is plenty of room for improvement of the system in both its usability and security.

In terms of usability, the system should be integrated with MIT's WebMoira to allow organizers to easily and quickly define voters by entering a WebMoira list that all members are in. This would greatly minimize the time required to start an election as most groups already have mail lists for their members. Additionally, a more friendly UI with text explaining the security behind the system and why the extra steps are required would increase voter participation and trust in the system.

While the main goal of this system is to be secure, there is still more work that could be done to achieve better security guarantees. This system relies heavily on the use of the administrator's private key in order to validate anonymous ballots. Currently, keys are created for each election and stored on the server's file system. An attacker with full access to the system could potentially compromise this key and render the election useless.

It should be noted however, that even if an attacker gains access to the private key, they will only be able to invalidate the results of the election rather than maliciously modify/influence them. Individual voters and third parties would be able to detect these fraudulent ballots by cross checking the admin's signed list, with the public bulletin board and the number of voters. A more secure key store, such as ones offered by Tyfone's secure card [4] would mitigate this issue by guaranteeing the security of keys on the server (i.e., once a key is created on a secure card, it can never be exported).

Another desirable property of voting systems is the prevention of vote selling. Vote selling would allow a corrupt party to influence the election by paying voters for voting a certain way. As it currently stands, this system (and any of the systems mentioned in the introduction) are not secure against vote selling. While at the end of the election, the system has no way of tying an individual ballot to a voter, the voting phase itself is open. This allows a corrupt party to witness the voting process and pay voters who vote a certain way. This issue was purposely not addressed as fixing it would require a closed voting process which could greatly inconvenience voters.

Lastly, while this system is open with the information it receives from its users (i.e., almost everything on the database is public), it is still a centralized system hosted on a single machine that could potentially be compromised. To deal with this issue, it is worth exploring the possibility of creating a decentralized version of this voting system. One may be able to use tools such as Blockstack [3] to run a server less application. The admin key could be kept secret using an implementation of Shamir's secret sharing algorithm and votes would be analogous to transactions in a bitcoin-like blockchain.

V. Conclusion

Most voting systems at MIT lack the basic security properties of a desirable voting system such as privacy and verifiability. I have implemented a cryptographically secure voting system tailored to the MIT community that provides, among other things, full voter anonymity and individual and third party election verifiability. This system is secure against internal and external malicious parties. Assuming the security of the cryptographic properties used, no party, including the election organizer will be able to falsely modify the results of the election.

Acknowledgments

I would like to thank Dorothy Curtis for her guidance and supervision throughout this project. Additionally, I would like to thank Ron Rivest for his excellent list of research papers regarding the topic of secure voting systems [6].

Availability

The server and client code is available in its entirety under the MIT license at https://github.com/Fertogo/elections. Deploy instructions are included in the README.

References

[1] Fujioka, A., Okamoto, T., Ohta, K. (n.d.). A Practical Secret Voting Scheme for Large Scale Elections. Retrieved from https://people.csail.mit.edu/rivest/voting/papers/FujiokaOkamotoOhta-APracticalSecretVotingSchemeForLargeScaleElections.pdf

[2] Dingledine, R., Mathewson, N., Syverson, P. (n.d.). Tor: The Second-Generation Onion Router. https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf

[3] Ali, M., Nelson, J. (n.d.). Blockstack: A Global Naming and Storage System Secured by Blockchains. https://blockstack.org/blockstack.pdf

[4] Tyfone Inc. Side-X Secure Token. Retrieved from https://tyfone.com/products/side-x-digital-endpoint-security/

[5] Chaum, D. (1982). Blind Signatures for Untraceable payments. Retrieved from http://www.hit.bme.hu/ buttyan/courses/BMEVIHIM219/2009/ Chaum.BlindSigForPayment.1982.PDF

[6] Rivest, R. L. (n.d.). Voting Resources Page. Retrieved from https://people.csail.mit.edu/rivest/voting/

[7] Fazel-Rezai, V. (2016, January 13). mit-cert-auth. Retrieved from https://github.com/vfazel/mit-cert-auth

Appendix A

Python

```python
# Usage: python blind.py ballotText

from Crypto.PublicKey import RSA
from Crypto.Hash import SHA256
from random import SystemRandom
import sys

def saveToFile(filename, data):
    file = open(filename, 'w')
    file.write(data)
    file.close()
    print "Created file: " + filename

def readFromFile(filename):
    file = open(filename, 'r')
    data = file.readlines()
    file.close()
    return ''.join(data).replace(">","").replace("<","")

def blindMessage(msg):
    print "Blinding ballot..."

    pub= RSA.importKey(readFromFile('key.pub'))

    r = SystemRandom().randrange(pub.n >> 10, pub.n)
    saveToFile('r', str(r));

    hash = SHA256.new()
    hash.update(msg)
    msgDigest = hash.hexdigest()

    msg_blinded = pub.blind(msgDigest, r)
    msg_blinded = ''.join(x.encode('hex') for x in msg_blinded) #hex

    print "Blinded Ballot:", msg_blinded
    # User sends msg_blinded to administrator for signature

blindMessage(sys.argv[1])
```

Appendix B

Python

```python
# Usage python unblind.py adminSignature

from Crypto.PublicKey import RSA
from Crypto.Hash import SHA256
from random import SystemRandom
import sys

def readFromFile(filename):
    file = open(filename, 'r')
    data = file.readlines()
    file.close()
    return ''.join(data).replace(">","").replace("<","")

def unblindMessage(msg_blinded_signature):
    pub= RSA.importKey(readFromFile('key.pub'))

    r = (long) (readFromFile('r'))

    msg_signature = pub.unblind(msg_blinded_signature, r)

    print "Signature for original ballot:"
    print msg_signature
    # User submits msg_signature to counter

    verifyMessage(msg_signature)

def verifyMessage(msg_signature):
    msg = raw_input("What was the original ballot? ")

    hash = SHA256.new()
    hash.update(msg)
    msgDigest = hash.hexdigest()

    if (pub.verify(msgDigest, (msg_signature,))):
        print "Signature is good. Submit signature and original ballot to the counter"
    else:
        print "The signature is invalid. Something went wrong"

unblindMessage(long(sys.argv[1]))
```