Private Decentralized E-Voting

Pak Hay Chan[*], Heng Li[†], Jordan Ugalde[‡], Yoni Stoller[§]

May 13, 2015

Abstract

In this paper, we present a new electronic voting scheme that does not rely on a trusted third party at any point. The cryptographic primitives we use to implement our scheme include El Gamal encryption, Random Exponentiation, Pedersen Commitments, and Cramer's Witness Hiding Protocol.

1 Introduction

Electronic voting's popularity is due, in part, to its high degree of efficiency, ease of voting, and fraud prevention. Accordingly, the success of e-voting is highly dependent on the performance of e-voting machines and software.

Most e-voting schemes require a trusted counting server as a third party. Therefore, the security of the third party is extremely critical to the voting system. By attacking this server, an external adversary or dishonest insider could modify votes between decryption and tabulation, shifting results in favor of the attacker's preferred candidate[1]. In this paper, we present a system that does not depend on a trusted third party server and thus avoids

[*]pakhay@mit.edu
[†]hengli@mit.edu
[‡]jugalde@mit.edu
[§]ystoller@mit.edu

the aforementioned threats.

Our scheme has the following security properties:

- *Correctness.* Our scheme prevents cheating with very high probability.
- *Privacy.* The individual vote of a participant cannot be determined by the other voters, or by any third party[1].
- *Verifiability.* Every voter tallies the votes individually. This, combined with the prevention of cheating provides voters with a way of verifying the election's outcome.
- *Democracy.* Every eligible voter has a single vote.
- *Fairness.* No voting information will be released before the tallying stage.

We begin by giving a summary of the primitives used by our voting scheme in Section 2. In Section 3 we describe in detail how our system's voting process works. Following that, we discuss ways of extending our system for more general cases in Section 4. Finally, in Section 5 we summarize our original and extended voting schemes.

2 Preliminaries

In this section we describe the cryptographic primitives used throughout our scheme.

2.1 ElGamal Cryptosystem

ElGamal is an asymmetric encryption scheme built on top of the Diffie-Hellman key exchange.

[1]Except for the extreme case where all the voters except for one collude against a single voter

- *Gen(1^k):* Given an input 1^k, *Gen* provides (G, p, q, g), where p is a safe prime ($p = 2q+1$), $\mathbb{G}$ is the group of quadratic residues modulo p (the order of $\mathbb{G}$ is q), and g is a generator of $\mathbb{G}$. Choosing any random $x \in \mathbb{Z}_q$, followed by computing $y = g^x$, yields the public key

  $(\mathbb{G}, q, g, y)$, and the private key $(\mathbb{G}, q, g, x)$. The message space is $\mathbb{G}$.

- *Enc(m):* Given a public key $p_k = (\mathbb{G}, q, g, y)$, and a message $m \in \mathbb{G}$, its encryption is $(c_1, c_2) = (g^s, y^s m)$, where s is a random, secret element chosen from $\mathbb{Z}_q$.

- *Dec(c):* Given a private key $s_k = (\mathbb{G}, q, g, x)$ and $(c1, c2)$, the decryption is $m = c_2/c_1^x = c_2 c_1^{-x}$

In our scheme, votes are represented as one of two numbers from the subset $\{1, v\}$, where v is a fixed generator of $\mathbb{G}$. The tallied vote is simply the product of all individual votes. Our scheme requires that the number of voters does not exceed $|\mathbb{G}|$.
Note that the homomorphic property of ElGamal guarantees that multiplying $Enc(v)$ by any element within $\mathbb{G}$ is equivalent to multiplying v by the same element.

2.2 Random Exponentiation

Our scheme uses Random Exponentiation (explained in greater detail in [2]) to provide participants with a way of decrypting the result of the group's tallied votes, without risking the exposure of any participant's individual vote.
Given an encryption C, each participant i chooses a secret $r_i \in \mathbb{Z}_q$ (different from the secret s_i of the participant's message) and computes C^{r_i}.
The participants then exchange their respective C^{r_i} values and multiply them to get $C_{\text{TOTAL}} = \prod C^{r_i} = C^{\sum r_i}$, i.e., C raised to the power of the sum of all the secret exponents.

Note that while $\sum r_i$ is known to all participants, each individual participant only knows his/her own secret r_i, thereby allowing us to raise C to an unknown power.

In this manner, participants can release $(y^{s_i r_i}, y^{r_i})$[2] to allow the decryption of C_{TOTAL}[3]. However, as stated above, since the underlying r_i values remain unknown, dishonest participants cannot use the information to compute y^{-s_i}, which would have allowed them to decrypt the vote of voter i (i.e., in order to calculate m_i, one must know y^{s_i}. Knowing $y^{s_i r_i}$ does not compromise the secrecy of y^{s_i}).

2.3 Pedersen Commitments

We use Pedersen commitments [3] as a way of making all participants commit to the $(y^{s_i r_i}, y^{r_i})$ values that they publish (see the explanation regarding Random Exponentiation) **before** viewing the values published by other participants. If we were to instead allow participants to reveal the information sequentially, the final participant could release a forged $(y^{s_i r_i}, y^{r_i})$, such that when multiplied with the other values, it would produce any element of the participant's choice in $\mathbb{G}$, thus compromising the validity of the voting system.

2.4 Cramer's Witness Hiding Protocol

Witness hiding (WH) is a weaker requirement than Zero-knowledge. Generally speaking, an interactive proof protocol is WH if the verifier cannot compute any new witness after engaging in the protocol [4]. Cramer et al. [5] constructed a simplified and efficient design of the WH protocol given that there exists a proof of knowledge protocol P that satisfies some basic requirements.

An extension of Cramer's protocol is verification of a ciphertext. We can

[2]Contrast with $(g^{s_i}, y^{s_i} m_i)$, the original message

[3]This is a slight simplification, as this only represents the first stage of the decryption. Random Exponentiation requires a total of $n + 1$ steps, where n is the number of voters

use the extended version of Cramer's protocol to prove that a ciphertext c encrypts one of the possible values $\{m_1, m_2, \dots\}$, without revealing which message c corresponds to.

Consider the case where the message space consists of two possible values. Then we have a simple protocol in which the prover commits to two pairs of ciphertexts. He claims that the two pairs are encryptions of the two permitted values. The verifier chooses one to open and the prover claims the other pair is valid. This protocol has soundness error of $1/2$, but by the extension of Cramer's work, we have a protocol with soundness error of 2^{-t} where t is the number of pairs of ciphertexts used in the protocol.

We will use this extension to verify that voters can only vote for one of the options within the message space.

3 Our Scheme

Our voting scheme has been designed to provide complete privacy to its voters. Although our design can be extended to larger parties, it is ideally suited for smaller groups. As such, voting is performed under the following circumstances:

- Every voter posts his/her vote on a website they all have access to. The website does not need to be trusted - it serves merely for convenience, as will be seen. The method for registering voters is not part of our design (assuming the group is small enough, such as a board committee, the voters can manually verify the identities of the voters).
- Every voter performs the tallying individually after all the required information is publicized. This allows each user to verify the outcome of the election.

Applying these conditions provides voters with a way of verifying that their votes were not discarded, and it also eliminates the need for a trusted third party.

3.1 Voting

Our scheme supports voting for one of two options. Let n be the number of voters. The voting stage consists of two parts: Encryption and Verification.

3.1.1 Encryption

First, the voters agree on a finite field $\mathbb{G}$ and a generator g of $\mathbb{G}$. Let q be the order of $\mathbb{G}$.

Every voter i then generates a public-private key pair by randomly chosing an $x_i \in G$. Then $(\mathbb{G}, q, g, y_i)$ and $(\mathbb{G}, q, g, x_i)$, where $y_i = g^{x_i}$, is a public-private key pair of ElGamal encryption, as described in section 2.1. A voter will use his private key to encrypt his vote.

The message space M is comprised of the two possible voting choices. Let $M = \{1, v\}$, where v is a generator of $\mathbb{G}$. Consider a message $m \in M$, if $m = 1$ represents the first voting option, and $m = v$ represents the second option.
Every voter $i = 1, 2, \ldots n$ has a vote $m_i \in M$ and encrypts the vote as follows:

$$\mathrm{Enc}_i(m_i) = (g^{s_i}, m_i \cdot y_i^{s_i})$$

where s_i is a random number chosen uniformly at random in $\mathbb{G}$ by each voter.

Thus, every encrypted vote has the form $(\alpha_i, m_i \cdot \beta_i)$ where $\alpha_i = g^{s_i}$ and $\beta_i = y_i^{s_i}$.

3.1.2 Verification

In order to guarantee that encrypted votes are valid, i.e., that the messages encrypted by the voters belong to the message space, we need to verify the votes.

This problem can be described as follows: Given the ciphertext of a vote, voters need to prove to all other voters that the ciphertext is an encryption of either $m = 1$ or $m = v$. Cramer's Witness Hiding Protocol, described in section 2.4 solves this problem. Thus we have a public procedure that guarantees that the probability of an individual voter cheating successfully (i.e, encrypts a messages not belonging to the message space without being discovered) is exponentially small.

After the votes are verified, the voters can start tallying.

3.2 Tallying

To tally the votes, we take advantage of the multiplicative homomorphic property of the ElGamal encryption.

Recall that each encrypted vote has the form $(\alpha_i, m_i \cdot \beta_i)$, where we multiply the second half$(m_i\beta_i)$ of all the encrypted votes.
Denote the product of all the encrypted votes as C:

$$\begin{aligned} C &= \Pi_{i=1}^{n} m_i \Pi_{i=1}^{n} \beta_i \\ &= v^{\#\text{ votes for the second option}} \Pi_{i=1}^{n} \beta_i \end{aligned}$$

since for those i for which $m_i = 1$, the term is dissolved in the product.
Let M denote $\prod m_i = v^{\#\text{ votes for second option}}$, and B denote $\prod \beta_i$, so $C = MB$.
To determine how many votes were cast for each option, it is sufficient to calculate how many votes were given to the second option. Our goal is thus to determine the value of M. The straightforward way of doing this would be to multiply C by the inverse of B, to reveal M. That is, $CB^{-1} = MBB^{-1} = M$. After discovering the value of M, the users could check for which value of $0 < k < \#voters,\ M = v^k$, where the k for which this is true represents the amount of voters for the second option.
The problem with this approach is that it requires the voters to release y^{s_i}, which would effectively allow participants to decrypt each others votes.

Instead, we will use Random Exponentiation to discover the result of the

election. First, note that the number of votes cast for the second option can be any integer value between 0 and the total number of voters. Let k represent a possible value. Since M = $v^{\text{\#votes for second option}}$, if k= #votes for the second option, then M times the inverse of $v^k = 1$.
However, if $k \neq$ #votes for second option, then M times the inverse of $v^k \neq 1$.

Let $P = v^{-k}C = v^{-k}MB$.
As explained above, Random Exponentiation allows a group of participants to raise a commonly known value to an unknown power, R. Given the value P, the group of voters can use Random Exponentiation to compute P^R, which is equal to $(v^{-k}M)^R B^R$. Likewise, the group of voters can jointly compute $(B^R)^{-1}$.

Thus, having calculated P^R and $(B^R)^{-1}$, the participants can compute $P^R(B^R)^{-1} = (v^{-k}M)^R B^R (B^R)^{-1} = (v^{-k}M)^R$.
Based on the value of $(v^{-k}M)^R$, the participants can conclude whether or not k = # votes for second option:

1. If $(v^{-k}M)^R = 1$, this implies $v^{-k} = M^{-1}$, and that k = # votes for the second option.
2. If $(v^{-k}M)^R \neq 1$, this implies $v^{-k} \neq M^{-1}$, and that $k \neq$ # voter for the second option.

Using this procedure, the voters can examine each possible value of k until finding one that gives $(v^{-k}M)^R = 1$, upon which the users will know that the number of votes cast for the second option was k.

The algorithm for doing this would look as follows:

```
foreach k from 0 to #voters:
        Compute $P = v^{-k}MB$
        Using Random Exponentiation, compute $P^R$, $(B^R)^{-1}$
        if $P^R \cdot (B^R)^{-1}$ == 1:
                return k
```

Note that each participant runs this algorithm separately (while exchanging values for Random Exponentiation), allowing them to verify that the result of the election is valid.

3.2.1 Simultaneous release of information

While Random Exponentiation provides us with a way of calculating $\left(B^R\right)^{-1}$ by having the participants release $(y^{s_i r_i}, y^{r_i})$, it does not specify the order in which they do so.
If a voter were to delay the release until after seeing all other values, he could privately compute the result of the election, and then release a tweaked value that would alter the result of the election. For example, if at the $k = 4$ round of the tallying $P^R \cdot \left(B^R\right)^{-1} = 1$, which implies that four voters voted for the second option, the voter could tweak his value before releasing it (e.g., by multiplying it by some random number larger than 1), and thus fool the other voters into thinking that the number of voters was not k, since $P^R \cdot \left(B^R\right)^{-1}$ would no longer equal 1.
In order to prevent this, we require all voters to release a Pedersen Commitment of their $(y^{s_i r_i}, y^{r_i})$ values. The values are only revealed after all voters have released their committed values.

3.2.2 Malicious voters and mathematical flukes

Although $P^R \cdot \left(B^R\right)^{-1}$ will always equal 1 for $v^{-k} = M^{-1}$, it will also equal 1 when $R = p - 1$. The probability of this happening is $\frac{1}{p-1}$, so this is not a major concern, since p must be very large in order to guarantee the security of ElGamal to begin with (it is also possible to simply require that the voters calculate $P^R \cdot \left(B^R\right)^{-1}$ for all possible values of k, and if $P^R \cdot \left(B^R\right)^{-1} = 1$ for more than one value of k, they repeat the election).
Likewise, there is always a chance that a dishonest voter will attempt to release a tweaked $(y^{s_i r_i}, y^{r_i})$, to cause $P^R \cdot \left(B^R\right)^{-1} = 1$. However, since the voter has no knowledge about the values that have been released (until after

he has committed), he is forced to guess, and as such, the probability of his success is also $\frac{1}{p-1}$.
That being said, a malicious voter who merely wishes to prevent the other voters from determining the result of the election may still do so by releasing a tweaked $(y^{s_i r_i}, y^{r_i})$ at every round (i.e., multiplying it by any value greater than 1). This will prevent $P^R \cdot (B^R)^{-1}$ from equaling 1 even for the correct value of k.
This aspect of the election prevents it from being suitable for certain applications. However, in the context of a smaller group, such as a committee board, we consider the cooperation of the participants to be a reasonable assumption.

4 Possible Extension

As presented thus far our voting system's primary limitations are the fact that there can only be two options to choose from and the fact that the voting scheme is a two stage process, so the absence of one voter in the second stage can make tallying impossible, requiring the whole process to start over.

4.1 Voting for Multiple Candidates

Referring back to Sections 2.1 and 3.2, any vote that is not a 1 must be a generator. However, if we allow multiple generators as votes, then it becomes ambiguous as to what value the result is a power of. For example, if we have $\mathbb{Z}_7^*$ then 2 and 4 are both generators of the group of quadratic residues mod 7, but since 4 is both 2^2 and 4^1 in $\mathbb{Z}_7^*$ it would be impossible to differentiate whether a value of 4 resulted from a single vote for 4 or two votes for 2. To address this limitation, we modify the voting system such that, rather than casting a single vote, the entire voting process is repeated for each candidate. That is, for each candidate, one either votes *for* the candidate or *against* the candidate. This sacrifices none of the security properties of our original scheme, but enables voting for multiple candidates. Furthermore, since the vote count for every candidate is known, we have the ability to choose the top k candidates who received the most votes. The primary issue with this extension is that it allows voters to vote for more than one candidate, so in situations where that is not desirable this extension should not be used. Since all votes for different candidates can be contained in one message to the server, there is no added complexity in terms of the number of messages that need to be passed. However, the Random Exponentiation phase still must happen for every candidate, so the time to process votes grows linearly with the number of candidates.

4.2 Allowing for Disappearing Voters

If the entire process must be restarted whenever a voter leaves during the election, then this system as stated would not be scalable. Our solution to this issue is, in large elections, to split the voters into subgroups of a predefined size, where each subgroup runs the election in parallel (after which the results of the subgroups' tallies would be combined). Then, if one voter leaves in the middle of the election, only the subgroup with the absent voter would have to restart the voting process, which drastically reduces the number of people that have to recompute the tallying phase[4]. Additionally, since Random Exponentiation has $n+1$ steps, using a smaller group size is far more efficient. This still has the undesirable quality that a single participant can stop the vote, by sending an invalid key in the second phase of the voting process. However, the stalling is encapsulated to just the subgroup containing the obstructing participant.

5 Conclusion

In this paper, we presented a scheme for voting between two possible options, which shares the security guarantees of other voting systems while also not

[4]It is worth noting that technically, user involvement is only required for the actual voting at the beginning of the election. Everything else can be automated.

requiring a trusted third party. Additionally, if one wishes to extend the system to allow for additional candidates, then a simple extension exists as long as one does not mind allowing voters to vote for multiple candidates. Finally, scaling this system to a large number of voters is accomplished by breaking down the voters into smaller subgroups such that if a vote needs to be recast, only the subgroup in question has to recompute the tallying.

References

[1] Ben Adida and Ronald L. Rivest. "Scratch & vote: self-contained paper-based cryptographic voting." WPES '06 (Alexandria, Virginia, 2006) pp. 29–39.

[2] Brandt, Felix. "Efficient cryptographic protocol design based on distributed El Gamal encryption." In Information Security and Cryptology-ICISC 2005, pp. 32-47. Springer Berlin Heidelberg, 2006.

[3] Pedersen, Torben Pryds. "Non-interactive and information-theoretic secure verifiable secret sharing." In Advances in CryptologyCRYPTO91, pp. 129-140. Springer Berlin Heidelberg, 1992.

[4] U. Feige and A. Shamir. . "Witness Indistinguishable and Witness Hiding Protocols, Proc. of STOC 90."

[5] Ronald Cramer, Ivan Damgard, and Berry Schoenmakers."Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols." In Yvo Desmedt, editor, CRYPTO, volume 839 of Lecture Notes in Computer Science, pages 174187. Springer, 1994.

[6] Fouard, Laure, Mathilde Duclos, and Pascal Lafourcade. "Survey on electronic voting schemes." Supported by the ANR Project AVOT (2007).