An optimally efficient multi-authority election scheme

Rosario Gennaro

Laboratory for Computer Science
Massachusetts Institute of Technology
545 Technology Square, Cambridge MA 02139
rosario@theory.lcs.mit.edu

Abstract

In this paper we present a new multi-authority election scheme. We achieve optimal efficiency for the voter compared to traditional multi-authority protocols. The voter simply posts a *single* encrypted message and a proof that such message encrypts a valid vote. This compares favorably with previous protocols in which the voter was required to carry on a secret sharing of her vote and encrypt a number of messages proportional to the number of authorities in the system.

One of the main tools of the paper is the notion of fault-tolerant threshold cryptosystems. These are protocols that allow to share the decryption power among a set of centers in a way that single messages can be decrypted without reconstruction of the private key. A bonus of our result is the construction of such a scheme for the ElGamal cryptosystems which we believe to be of its own interest.

Note: This paper has not been presented at and is not currently submitted to any conference.

1 Introduction

Electronic election protocols have been very popular in the crypto community in the past ten years. It is probably one of the most important application of secure multi-party computation. Yet despite extensive work on the subject, there is still room for improvement on the efficiency of protocols and on their robustness against stronger kind of adversaries.

When talking about efficiency it should be clear that the main consideration must be given to the computational effort required from a single voter. Indeed while governments can (and do nowadays) afford a large organizational effort to hold elections (including a large computational effort), it is mandatory to make the voting task as efficient as possible for the voter who might be participating from home with his PC.

In this paper we present a simple multi-authority election scheme in which the task of the voter is reduced to the bare minimum. The voter simply posts a *single* encrypted message and a proof that such message encrypts a valid vote. This compares favorably with previous protocols in which the voter was required to carry on a secret sharing of her vote among the voting centers, post several encrypted messages (one for each center) and a produce a proof of correctness of the vote that (because of such sharing) would be more complicated and longer than ours. Our protocol results in much lower computational and communication cost for the voter.

1.1 Previous work

Election protocols have been studied extensively in the cryptographic literature. General solutions based on general secure multi-party computation can be found in [27, 2, 14, 34, 9] but they are hopelessly inefficient.

Very efficient solutions based on anonymous channels (or mixers) were proposed in [11, 22, 31, 36]. In these schemes (as in ours) the voter posts a single message, but the protocols all suffer from a lack of fault-tolerance. Indeed the anonymous channels are based on a cascade of servers that mix the identity of the voters. The crash of even one of these servers would result in the failure of the whole protocol.

Benaloh in his Ph.D. thesis [6] proposes what has become a model for multi-authority elections. He introduces homomorphic encryption functions and secret sharing techniques to achieve greater efficiency, fault-tolerance and universal verifiability (i.e. later auditing of the correctness of the elections by external parties.) Other schemes based on Benaloh's approach are [4, 35, 16]. Our scheme will be modeled after Benaloh's approach so we will see these techniques in detail later in the paper.

COMPLEXITY ISSUES. If we let n to be the number of the voting authorities and k a security parameter in Benaloh's original solution each voter was required to transmit $O(nk^2)$ bits and perform $O(nk^2)$ elementary operations. The solution by Cramer *et al.* in [16] improved this to $O(nk)$. Independently, in a previous version of this paper [25], we had discovered a way to reduce the complexity of Benaloh's approach to $O(k^2)$.

1.2 This paper's improvements

In this paper we combine the techniques of [16] and [25] to obtain an election protocol whose voter complexity (both computational and communication) is $O(k)$. This is clearly optimal.

In schemes that follow Benaloh's approach [4, 35, 16] the voter is required to share his vote among the centers using Shamir's secret sharing, send such shares encrypted with the centers public key and finally prove the correctness of the sharing and of the vote. The sharing will protect the privacy of the voter even if up to a fixed threshold of the centers cooperate to learn the casted vote. The main contribution of [16] was to show how to perform the correctness proof in an extremely efficient way.

We further reduce the complexity by introducing a new tool: *fault-tolerant threshold cryptosystems*. In our case there will be only one public-key whose matching secret-key is shared among the centers using threshold cryptography techniques (see [18] for a survey.) The voter casts his vote encrypted and produces a proof of correctness. The privacy of the vote is still preserved since nobody knows the secret key, but it is required the cooperation of a fixed threshold of the center in order to decrypt. In order to assure the correctness of the shared decryption procedure we need to make sure that it is *robust*, i.e. reaches a correct result even in the presence of maliciously behaving centers. A bonus of our paper is the construction of such a scheme for the ElGamal cryptosystem which we believe to be of its own interest.

The resulting protocol has three major properties. Let n be the number of voting authorities.

1. the work required to the voter is minimal. In particular the number of public-key operations is reduced by a multiplicative factor of n. Although n is usually much smaller than k, this is still a substantial gain in practice.

2. the protocol for the user remains the same even if n is variable. Usually n grows with the desired security of the scheme (the more centers the less potential than an adversary can corrupt half of them). Using our protocol this growth is "transparent" to the user. This means that the software for the election protocol can be distributed once and for all and then the value n can be fixed a posteriori.

3. Using recent techniques on proactive threshold cryptosystems [1] it is possible to leave the system (and its keys) in place for a really long time without fearing that the secret key gets compromised. More details in Section 6

As a final remark we would like to point out that the improved voter-efficiency does not come at the expenses of the efficiency of the protocol for the centers. Indeed the communication and computational complexity required from the centers in our protocol is comparable to the one in [16] (more specifically $O(mk)$ where m is the number of voters.)

1.3 Incoercibility

Recently Benaloh and Tuinstra in [5] introduced the notion of *receipt* for electronic election schemes. They showed how previous election protocols in the literature all suffer from a common defect: a voter can carry away from the protocol a receipt that proves the way he voted. This receipt can then be used for vote buying or for coercion (indeed the voter may be required to save such receipt under some physical threat). Obviously this defect makes previous schemes unusable in practice. It is important to notice that this is an inherent problem of *all* multi-party computation protocol known so far [27, 2, 14, 34, 9].

In their paper Benaloh and Tuinstra introduced the notion of *receipt-free* or *incoercible* election scheme. Such schemes allow a voter at the end of the protocol to lie about the casted vote even if under coercion. Indeed given the execution of the protocol the voter will be able to create two different histories of his computations that are both consistent with such execution while corresponding to two different casted votes. Other receipt-free schemes were presented in [36, 30]. All these schemes rely on some form of physical assumption to obtain the desired Incoercibility. Recently Gennaro and Canetti showed that it is possible to perform a multiparty computation of any function in an incoercible fashion *without* relying on physical assumption [10]. In particular their result solves the problem for the case of election protocols.

We are not going to address incoercibilty in this paper.

1.4 Structure of the paper

In Section 2 we present the new tool of fault-tolerant threshold cryptosystems. In Section 3 we talk about homomorphic encryption schemes and their use in election protocols. Section 4 describes our protocol using a generic cryptosystem. We instantiate the latter to a specific number-theoretic scheme in Section 5. In Section 6 we discuss the properties of our protocol.

2 Fault-tolerant Threshold Cryptosystems

In this section we will introduce the abstraction of fault-tolerant threshold cryptosystems. In section 5 we will show how to implement them using standard number-theoretic assumptions.

A probabilistic encryption scheme $\mathcal{E}$ is a triplet of efficient algorithms $(\mathcal{KGA}, \mathcal{EA}, \mathcal{DA})$. $\mathcal{KGA}$ is the key-generation algorithm. On input a random string ρ, it outputs a pair (E, D) where E is called the public-key and D the secret-key. $\mathcal{EA}$ is the encryption algorithm. On input E, a message v and a random string r it outputs the ciphertext c which we denote with $E_r(v)$. We say that $\mathcal{E}$ is secure if, for any two messages v_1 and v_2, the probability distributions $\{E_r(v_1)\}_r$ and $\{E_s(v_2)\}_s$ are computationally indistinguishable [28]. $\mathcal{DA}$ is the decryption algorithm. On input D, and a ciphertext $c = E_r(v)$ the output of $\mathcal{DA}$ is the message v.

A (t, n)-threshold cryptosystem $\mathcal{TE}$ for $\mathcal{E}$ is a pair of protocols $(\mathcal{KGP}, \mathcal{DP})$ for a set of centers $C_1, \ldots, C_n$. $\mathcal{KGP}$ is the key generation protocol. Each center C_i runs it on input a random number ρ_i. The public output is E and the private output of the center C_i is a value d_i. The d_i's constitute a (t, n)-threshold sharing of the value D such that $(E, D) = \mathcal{KGA}(\rho)$ and ρ is a random value. $\mathcal{DP}$ is the decryption protocol. Each center runs it on input a ciphertext $c = E_r(v)$ and the value d_i. The public output of the protocol is the value v.

We assume that there is an adversary that can corrupt up to $t - 1$ centers and make them deviate arbitrarily from their prescribed instructions. We say that $\mathcal{TE}$ is a fault-tolerant threshold cryptosystem if the two protocols $\mathcal{KGP}$ and $\mathcal{DP}$ are robust, that is they compute the correct output even in the presence of such adversary. We also require that the protocols $\mathcal{KGP}$ and $\mathcal{DP}$ are zero-knowledge for such an adversary. This will assure us that executions of the protocols will not give the adversary any advantage into trying to decrypt ciphertexts on her own.

Fault-tolerant threshold cryptosystems can be constructed for any encryption scheme $\mathcal{E}$ by using the generic results on secure multiparty computation [27]. However we will concentrate on efficient implementations of this primitive. In section 5 we will present one possible implementation based on the hardness of discrete-log.

We actually believe this to be the first full efficient implementation of a fault-tolerant threshold cryptosystem. Indeed RSA-based solutions have been proposed recently [21, 26], but they are only partial implementation as they provide only a fault-tolerant decryption protocol $\mathcal{DP}$. Such solutions rely on a trusted center to generate the keys. More details on our implementation and on other applications of fault-tolerant threshold cryptosystems appear in Appendix B.

3 Homomorphic Encryption Schemes and their use in election protocols

The idea of using homomorphic encryption schemes in the context of election protocols is due to Benaloh. For a complete overview of the techniques the reader is referred to [6].

Let $\mathcal{E}$ denote a probabilistic encryption scheme. Let M be the message space and C the ciphertext space. Let M be a group under the operation $\oplus$. Let C be a group under the operation $\otimes$. We say that $\mathcal{E}$ is a $(\oplus, \otimes)$-homomorphic encryption scheme if, for any public key E, given $c_1 = E_{r_1}(m_1)$ and $c_2 = E_{r_2}(m_2)$ there exist an r such that

$$c_1 \otimes c_2 = E_r(m_1 \oplus m_2)$$

Now let $\mathcal{E}$ be a $(+, \otimes)$ homomorphic scheme where $\otimes$ is an efficiently computable operation over the space of the ciphertexts. Benaloh's idea for an election protocol could be summarized in the following way. Assume that there are m voters $V_1, \ldots, V_m$ and n centers $C_1, \ldots, C_n$. Assume votes are in $\{0, 1\}$.

Set Up Each center C_j broadcasts an instance $E^{(j)}$ of the homomorphic scheme $\mathcal{E}$, keeping secret the corresponding secret key.

Vote Casting Let $v_i \in \{-1, 1\}$ be voter V_i's vote. V_i shares v_i using Shamir's (t, n)-threshold secret sharing protocol [38]. I.e., he chooses a random polynomial f_i over M of degree $t-1$ such that $f_i(0) = v_i$ and sets $s_{i,j} = f_i(j)$. Finally V_i sends $\sigma_{i,j} = E^{(j)}_{r_{i,j}}(s_{i,j})$ to center C_j.

Tally Counting Center C_j sets $c_j = \prod_i \sigma_{i,j}$. Because of the homomorphic properties of the encryption scheme $c_j = E^{(j)}_{r_j}(\sum_i s_{i,j})$. C_j reveals $t_j = \sum_i s_{i,j}$. Because of the homomorphic properties of Shamir's secret sharing scheme (see [3]) we have that the t_j's are a (t, n)-threshold sharing of the tally $T = \sum_i v_i$ which can be now reconstructed.

Notice that the secrecy of each vote is preserved against a coalition of $t - 1$ centers. The protocol above however has many problems. For example a voter can cast an irregular vote (for example 1000 instead or 1 or -1). Also the scheme is not *universally verifiable*, meaning that at a later time nobody cold check on the correctness of the tally by just looking at the transcript of the protocol. In [6] fixes are presented for these problems and we will recall these techniques in the next section when we describe our protocol.

4 Our generic election scheme

In this section we describe a "generic" implementation of our protocol. That is, we do not specify here the particular encryption function we are going to use and various other details. In Section 5 we will fill in the missing details.

Let $\mathcal{E}$ be a homomorphic encryption scheme (Section 3) and let $\mathcal{TE} = (\mathcal{KGP}, \mathcal{DP})$ be an efficient fault-tolerant (t, n)-threshold cryptosystem for $\mathcal{E}$ (Section 2). Our protocol works as follows.

Phase 1: Setting Up. In the first phase of the protocol the centers set up the encryption function for the protocol, by generating together an instance of $\mathcal{E}$. To do so they execute protocol $\mathcal{KGP}$. The public output will be E and the private output of center C_j will be d_j.

Phase 2: Vote Casting. In order to cast his vote $v_i \in \{0, 1\}$, each voter V_i posts an encrypted message $c_i = E_{r_i}(v_i)$.

Phase 3: Vote Checking. The centers want to make sure that each voter has casted a correct vote, i.e. either a 1 or a -1. They require the voter V_i to engage in a zero-knowledge proof that the casted vote is a correct one.

Here one could use the "cryptographic capsules" technique of Benaloh [7], but instead we will use the more efficient proof from [16].

Phase 4: Tally Counting. Let $c = \prod_{i=1}^{m} c_i$. Because of the homomorphic properties of $\mathcal{E}$ we have that $c = E_r(T)$ where $T = \sum_i v_i$ is the tally. The centers can compute T using the shared decryption protocol $\mathcal{DP}$.

Properties: Notice that the privacy of the individual votes is protected against a coalition of $t - 1$ centers as in Benaloh's scheme. Indeed in order to decrypt anything one needs at least t centers, given the properties of threshold cryptosystems. Notice also that the fault-tolerance of $\mathcal{TE}$ assures us that the result of the tally computation is correct under the assumption that there are at least t malicious centers. The scheme is not yet universally verifiable. Once we instantiate the encryption scheme $\mathcal{E}$ we will show how such property can be obtained though.

Efficiency: Notice how the protocol is much more efficient for the voter. Compared to Benaloh-like schemes (Section 3) with one simple move we have reduced the number of public-key operations that each voter has to perform by a multiplicative factor of n. This does *not* come with an unbearable increase of the computation required by the centers. Indeed in the Benaloh-like schemes the centers perform one decryption. Here (Phase 4) they simply participate in $\mathcal{DP}$ once. If the threshold scheme is efficient (as the one presented later by us) this participation requires only few public-key operations.

5 Choosing the encryption function

In this section we will instantiate the encryption scheme $\mathcal{E}$ to a specific number-theoretic implementations. We need that $\mathcal{E}$ be homomorphic and that an efficient fault-tolerant threshold cryptosystem based on $\mathcal{E}$ exists.

r-RESIDUOSITY. Benaloh in [6] proposed an homomorphic encryption scheme based on factoring. It is an extension of the quadratic residuosity scheme of Goldwasser and Micali [28]. Unfortunately it is not known how to perform efficiently a distributed key generation protocol $\mathcal{KGP}$ for cryptosystems based on factoring. One solution is to perform secure multiparty computation [27] to initialize the system. But as we said earlier this solution is not acceptable for efficiency reasons. Another option would be to assume a trusted *dealer* that initializes the system by sharing the secret key among the centers. But this solution is also not good because this "dealer" would know the master secret key and be able to decrypt every single vote.

This is why in the next section we will present a different homomorphic encryption scheme based on the ElGamal cryptosystem [23].

5.1 A modification of ElGamal

Primes and discrete logs: If p is a prime, we denote with Z_p^* the multiplicative group modulo p. If $h \in Z_p^*$ we denote with $ord(h)$ the *order* of h that is the minimum integer $\omega \leq p-1$ such that $h^\omega = 1 \bmod p$. If $x \in Z_p^*$ with $DL_h(x)$ we denote the *discrete logarithm* of x in base y, that is the unique $0 \leq i \leq ord(y)$ such that $y^i = x \bmod p$. An element g is said to be a generator of Z_p^* if $ord(g) = p-1$.

We describe a variant of the ElGamal public–key encryption scheme [23]. Let p be a prime of the form $kq+1$ where q is another large prime. We assume that it is hard to compute discrete logs modulo q in Z_p^*. Our encryption scheme, that we will denote with MEG for Modified ElGamal, works as follows:

Public information: p, q and g an element of order q in Z_p^*. Let the message space be Z_q. Assume $q > m$ the number of voters.

Key generation: Choose a random x in Z_q and compute $y = g^x \bmod p$. Publish y as the public key and keep x as the secret key.

Encryption: Given a message v choose a random $r \bmod q$. Compute $\alpha = g^r \bmod p$ and $\beta = y^r g^v \bmod p$. The ciphertext will be

$$c = MEG_r(v) = (\alpha, \beta)$$

Decryption: Given $c = (\alpha, \beta)$ clearly

$$g^v = \frac{\beta}{\alpha} \bmod p$$

and thus

$$v = DL_g(\frac{\beta}{\alpha})$$

This is in general a hard computation to perform, but if we know that $v \leq T$ then we can use the baby-step giant-step algorithm to compute v in $O(\sqrt{T} \log T)$ operations using $O(\sqrt{T})$ storage (see section 3.1 in [29]). In our case T is the tally of the election so $T \leq m$ the number of voters [1]. The efficiency of the decryption algorithm for this scheme is thus comparable to Benaloh's r-residuosity scheme (see [6]). Notice that, because of the randomization of ElGamal encryption, this bound on the decryption time is valid only for the holder of the secret key x.

Homomorphic property: Given two ciphertexts $c_1 = (\alpha_1, \beta_1)$ and $c_2 = (\alpha_2, \beta_2)$ we can define

$$c_1 \otimes c_2 = (\alpha_1 \alpha_2 \bmod p, \beta_1 \beta_2 \bmod p)$$

as the component wise multiplication of the two ciphertexts. MEG is a $(+, \otimes)$ homomorphic encryption scheme. Indeed if $c_1 = MEG_{r_1}(m_1)$ and $c_2 = MEG_{r_2}(m_2)$ then

$$c_1 \otimes c_2 = MEG_{r_1 + r_2 \bmod q}(m_1 + m_2 \bmod q)$$

Notice this means that during Phase 4 we are actually computing $T \bmod q$, but since $q > m$, this is the real tally.

5.2 Distributing the power of decryption

Now we will describe a fault-tolerant (t, n)-threshold cryptosystem for MEG with $n \geq 2t+1$, that is we need a simple majority of honest players (clearly optimal). Although we present it here only for MEG it is very easy to modify the scheme so that it works also for the traditional ElGamal scheme. Since we believe the scheme to be of its own interest we present it in Appendix B.

The basic idea of this protocol is due to Desmedt and Frankel [17], however in their case a trusted party had to generate the keys and malicious behavior of the centers was not tolerable at decryption time. Pedersen [32] (using work of Feldman [19]) shows how to eliminate the trusted dealer for the sharing of the key. We add fault-tolerance also to the decryption phase. In order to do so we need to modify Pedersen's protocol in order to tolerate malicious centers and we also adapt techniques on undeniable signatures due to Chaum ([13, 8]).

Key generation protocol $\mathcal{KGP}$: The centers have to agree on the parameters of the system (p, q, g). One possible way of doing this is to have one of them (say C_1) generate them and

[1] Actually $-\frac{m}{2} \leq T \leq \frac{m}{2}$ but it is possible to "normalize" the interval back to $0 \leq T \leq m$, by multiplying the final ciphertext by $g^{\frac{m}{2}}$

the rest check that they are correct (it is easy to check if C_1 gives the complete factorization of $p-1$.) As said above the actual key generation protocol is due to Pedersen [32]. For sake of completeness we report it anyway.

1. Each center C_i generates a random number $S_i \bmod q$ and a random polynomial

$$f_i(Z) = \sum_{k=1}^{t} a_{ik} Z^k \bmod q$$

of degree t and such that $f_i(0) = a_{i0} = S_i$.

2. C_i then uses Feldman's Verifiable Secret Sharing protocol [19] to share S_i among the other centers. That is C_i sends to C_j privately the value $s_{ij} = f_i(j) \bmod q$ and then publishes the values $y_{ik} = g^{a_{ik}} \bmod p$.

3. Now each center C_j can verify he received a good share. Indeed since

$$s_{ij} = f_i(j) = \sum_{k=1}^{t} a_{ik} j^k \bmod q$$

then it must be that

$$g^{s_{ij}} = \prod_{k=1}^{t} y_{ik}^{j^k} \bmod p$$

If the above verification equation fails, C_j casts a complaint against C_i

4. If more than t complaints are filed against C_i then he is disqualified. Otherwise for each complaining C_j, C_i will reveal s_{ij} matching the verification equation. If he does not comply to this step C_i is disqualified. Let G be the set of the indices of the non disqualified centers.

5. Center C_i now set

$$x_i = \sum_{j \in G} s_{ji} \bmod q$$

Clearly this is C_i's share of the value

$$x = \sum_{j \in G} S_j \bmod q$$

for a (t, n)-threshold scheme. x is going to be the secret key of the group.

6. The public key is set as

$$y = \prod_{j \in G} y_{j0} \bmod p$$

Clearly (y, x) is a legitimate public–key, secret–key pair since $y_{j0} = g^{S_j}$ and so

$$y = \prod_{j \in G} g^{S_j} = g^{\sum S_j \bmod q} = g^x$$

What one needs to remember about this protocol is that each center C_i has now a share x_i for a Shamir-based (t, n)-threshold scheme of the private key x. The matching key y is public. This means that if Λ is a subset of $t+1$ centers, x can be computed as

$$x = \sum_{i \in \Lambda} \lambda_{i,\Lambda} x_i \bmod q \tag{1}$$

where the $\lambda_{i,\Lambda}$ are the usual Lagrange interpolation coefficients. Also for each x_i the value $z_i = g^{x_i} \bmod p$ is made public by the protocol.

Decryption Protocol $\mathcal{DP}$: We saw above how to recompute the secret key x from the shares of t centers. However we don't want to recompute x at decryption time, but only decrypt a single ciphertext $c = MEG_r(v)$ without revealing the key. Also we want to make sure that this decryption is correct even if up to t centers are maliciously cooperating to disrupt the protocol.

1. Given a ciphertext $c = (\alpha, \beta) = MEG(v)$ each center C_i will broadcast w_i which is supposed to be α^{x_i}

2. In order to detect centers who are not broadcasting the correct w_i we require each center C_i to prove in zero–knowledge (using the protocol described in the Appendix A) that

$$DL_g(z_i) = DL_\alpha(w_i)$$

3. Let Λ a random subset of $t+1$ centers who passed the above zero–knowledge proof. Then we have

$$\alpha^x = \prod_{i \in \Lambda} (w_i)^{\lambda_{i,\Lambda}} \bmod p$$

   by raising α to both sides of equation (1).

4. Set $\hat{v} = \frac{\beta}{\alpha^x} \bmod p$

5. The original plaintext is $v = DL_g(\hat{v})$

5.3 Proof of vote correctness

We have been filling all the missing details about the protocol, except the proof of correctness of the vote. Given the encryption $c_i = MEG_{r_i}(v_i) = (\alpha_i, \beta_i)$ of the vote, we require voter V_i to prove that either $\beta_i = y^{r_i} g$ or $\beta_i = y^{r_i}/g$.

For this purpose we use the witness-indistinguishable protocol presented in [16] which in turn is based on a protocol in [15]. This is a simple three-move, public-coin interactive protocol that the voter runs as the prover, while the centers act collectively as the verifier. For simplicity let us drop the voter's index i.

Input for Prover and Verifier: A prime $p = kq + 1$, q, g element of order q in Z_p^*, $y = g^x \bmod p$, $\beta = y^r g^v$.
Input for the Prover: $r, v \in Z_q$
Task: Prove that $v = \pm 1$

Case $v = 1$

1. The Prover chooses $a_1, b_1, u_2 \in Z_q$ at random and sets:
   $A_1 \leftarrow y^{a_1}(\beta g)^{-b_1} \bmod p$
   $A_2 \leftarrow y^{u_2} \bmod p$
   Sends A_1, A_2 over to the Verifier.

2. The Verifier chooses $c \in Z_q$ at random and sends it to the Prover.

3. The Prover sets:
   $b_2 \leftarrow c - b_1 \bmod q$
   $a_2 \leftarrow u_2 + rb_2 \bmod q$
   Sends a_1, a_2, b_1, b_2 over to the Verifier.

4. The Verifier checks that:
   $b_1 + b_2 = c \bmod q$
   $g^{a_1} = A_1(\beta g)^{b_1} \bmod p$
   $g^{a_2} = A_2(\beta / g)^{b_2} \bmod p$

Case $v = -1$

1. The Prover chooses $a_2, b_2, u_1 \in Z_q$ at random and sets:
   $A_1 \leftarrow y^{u_1} \bmod p$
   $A_2 \leftarrow y^{a_2}(\beta / g)^{-b_2} \bmod p$
   Sends A_1, A_2 over to the Verifier.

2. The Verifier chooses $c \in Z_q$ at random and sends it to the Prover.

3. The Prover sets:
   $b_1 \leftarrow c - b_2 \bmod q$
   $a_1 \leftarrow u_1 + rb_1 \bmod q$
   Sends a_1, a_2, b_1, b_2 over to the Verifier.

4. The Verifier checks that:
   $b_1 + b_2 = c \bmod q$
   $g^{a_1} = A_1(\beta g)^{b_1} \bmod p$
   $g^{a_2} = A_2(\beta / g)^{b_2} \bmod p$

In [16] it is proven that the above protocol is a witness-indistinguishable proof that the vote is correct. Readers are referred to that paper for details on the proof.

6 Various Remarks on the protocol

Now that we have instantiate the encryption function (the only missing part in the protocol of Section 4) we can state some of its extra properties.

1. Removing interaction. Vote casting can be made non-interactive by assuming a beacon [33, 6] that broadcasts random bits at regular intervals. So during vote checking, the challenge

c could be determined by taking the random bits of the beacon at that time. A way of implementing this in practice is to use the Fiat–Shamir heuristic [20]. That is c could be computed by applying a suitable hash function H to the first-round messages of the voter. In order to avoid possible correlations among casted votes (see [24]) an identity string of the voter should be added to the argument of the hash function. With this improvement our protocol simulates real-life elections where the voter casts his vote and walks away without interacting with the voting authorities.

Similarly the centers can remove interaction in their zero-knowledge proofs during the decryption protocol by using the Fiat-Shamir heuristic (see Appendix A for more details.)

2. The protocol is universally verifiable. If we use the non–interactive version of the protocol (either with the beacon or with the Fiat-Shamir heuristic) then the election is universally verifiable since each step, from setting up to vote counting comes with an attached proof of correctness that can be publicly verified, as its randomness comes from a trusted source (the beacon or the hash function.)

3. Fault tolerance. Our protocol tolerates at most $t = n/2$ malicious faults among the centers. Since they have to provide zero-knowledge proofs that they are decrypting correctly, faulty centers will be detected ad eliminated. If there are less than t faulty centers the protocol will successfully complete. That is we achieve the same fault-tolerance of Benaloh-like schemes. This compares favorably with mix-channel based protocols [12, 22, 31, 36] which do detect faulty centers, but they are not able to tolerate them. Indeed in such protocols just one faulty center will cause the entire election to halt and to be repeated.

4. Efficiency. Although it should be clear to the reader at this point, we repeat that the computational and communication cost of vote-casting is now simply $O(k)$ where k is the security parameter. There is no dependence on n. This is clearly optimal. Notice that the cost of computing the tally and of an eventual auditing of the election protocol remains unchanged from the protocol in [16] to $O(nk)$.

Notice that we get the fault-tolerance of Benaloh-like schemes with an efficiency for the voter that is comparable to the one of mix-channels protocols.

4. Long lived keys. It is possible to use the same setup and the same public–secret key pair (y, x) for many elections. In this case however the secrecy of the key x could be compromised by a mobile attacker that in a long time could break into more than t centers. If keys are used for multiple elections then it is recommendable to refresh the shares using the techniques described in [1] for proactive public-key cryptosystems, which apply to ElGamal-like cryptosystems like ours.

7 Conclusions

We presented a simple multi-authority election protocol. The protocol has the desirable feature of being extremely voter-efficient. The protocol is based on a new tool that we call fault-tolerant threshold cryptosystems. A bonus of our result is the construction of a fault-tolerant threshold cryptosystem for the ElGamal scheme which we believe to be of its own interest.

References

[1] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10, Chicago, 1988. ACM.

[2] J. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*. Springer, 1987. Lecture Notes in Computer Science No. 263.

[3] J. Benaloh and M.Yung. Distributing the power of government to enhance the power of voters. In *Proc. 5th ACM Symp. on Principles of Distributed Computation*, pages 52–62. ACM, 1986.

[4] J. Benaloh and D. Tuinstra. Receipt-free secret ballot elections. In *26th ACM Symposium on Theory of Computing*, pages 544–553, 1994.

[5] Josh Benaloh. Verifiable secret ballot elections. Technical Report TR–561, Yale Department of Computer Science, September 1987.

[6] Josh Cohen Benaloh. Cryptographic capsules: A disjunctive primative for interactive protocols. In A.M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 213–222. Springer-Verlag, 1987. Lecture Notes in Computer Science No. 263.

[7] J. Boyar, D. Chaum, I. Damgård, and T. Pedersen. Convertible undeniable signatures. In A.J. Menezes and S. A. Vanstone, editors, *Proc. CRYPTO 90*, pages 189–205. Springer-Verlag, 1991. Lecture Notes in Computer Science No. 537.

[8] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multiparty computation. In *27th ACM Symposium on Theory of Computing*, pages 239–248, 1996.

[9] R. Canetti and R. Gennaro. Incoercible multiparty computation. In *37th IEEE Symposium on Foundations of Computer Science*, 1996. To appear.

[10] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24:84–88, February 1981.

[11] David Chaum. Blind signature system. In D. Chaum, editor, *Proc. CRYPTO 83*, pages 153–153, New York, 1984. Plenum Press.

[12] David Chaum. Zero–knowledge undeniable signatures. In *EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 458–464. Springer-Verlag, 1990.

[13] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 462–462. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[14] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo G. Desmedt, editor, *Proc. CRYPTO 95*, pages 174–187. Springer, 1994. Lecture Notes in Computer Science No. 839.

[15] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In *EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1996.

[16] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 307–315. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[17] Yvo G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.

[18] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc.* 28*th IEEE Symp. on Foundations of Comp. Science*, pages 427–438, Los Angeles, 1987. IEEE.

[19] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 186–194. Springer, 1987. Lecture Notes in Computer Science No. 263.

[20] Y. Frankel, P. Gemmell, and M. Yung. Witness-based Cryptographic Program Checking and Robust Function Sharing. Proceedings of STOC'96, 1996.

[21] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *AusCrypt'92*, pages 244–251, 1992.

[22] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

[23] Rosario Gennaro. Achieving independence efficiently and securely. In *Proc.* 14*th ACM Symp. on Principles of Distributed Computation*, pages 130–136. ACM, 1995.

[24] Rosario Gennaro. A multi-authority election protocol which is voter efficient and receipt-free against a dynamic coercer. Manuscript, 1996.

[25] Rosario Gennaro, Stas Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. To appear in CRYPTO 1996.

[26] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *19th ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[27] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[28] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public-key and signature schemes. Manuscript, 1995.

[29] A. K. Lenstra and H. W. Lenstra, Jr. Algorithms in number theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, chapter 12, pages 673–715. Elsevier and MIT Press, 1990.

[30] V. Niemi and A. Renvall. How to prevent buying of votes in computer elections. In *ASYACRYPT*, pages 141–148, 1994.

[31] C. Park, K. Itoh, and K. Kurosawa. All/nothing election scheme and anonymous channel. In *EuroCrypt'93*, 1993.

[32] T. Pedersen. Distributed provers with applications to undeniable signatures. In *EuroCrypt'91*, 1991.

[33] M. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, October 1983.

[34] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st ACM Symposium on Theory of Computing*, pages 73–85, 1989.

[35] Kazue Sako and Joe Kilian. Secure voting using partially compatible homomorphisms. In Yvo G. Desmedt, editor, *Proc. CRYPTO 95*, pages 411–424. Springer, 1994. Lecture Notes in Computer Science No. 839.

[36] Kazue Sako and Joe Kilian. Receipt-free mix-type voting schemes. a practical implementation of a voting booth. In *EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer-Verlag, 1995.

[37] Alredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 522–533, Santa Fe, 1994. IEEE.

[38] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.

A ZK proofs for equality of discrete logs

In this section we will describe two protocols to prove in zero–knowledge that two elements in Z_P^* have the same discrete logarithm with respect to two different bases.

Let P be a prime, g a generator for Z_P^* and h an element in Z_P^* (not necessarily a generator). Given two elements $y, z \in Z_P^*$ Alice wants to prove to Bob that there exists x such that

$$y = g^x \bmod P \text{ and } z = h^x \bmod P$$

without revealing anything else.

Protocol A: This protocol was invented by Chaum and appeared first in [13]. A proof of security can be found in [8].

1. Bob chooses integers a, b computes $c = g^a h^b \bmod P$ and sends c to Alice
2. Alice chooses a random t and sets $d_1 = cg^t$ and $d_2 = d_1^x$. She sends (d_1, d_2) to Bob
3. Bob reveals a and b
4. Alice checks that $c = g^a h^b \bmod P$ and if so reveals t
5. Bob checks that $d_1 = g^{a+t} h^b$ and $d_2 = y^{a+t} z^b$

Protocol B: This different protocol achieves the same goal using quite different means. We do not believe that this protocol has appeared anywhere else.

1. Alice chooses a random $r \bmod (P-1)$ and computes $d_1 = g^r$ and $d_2 = h^r$. Alice sends d_1 and d_2 to Bob.

2. Bob chooses a random bit b and sends it to Alice

3. Alice replies with $w = r + bx \bmod (P-1)$. That is, $w = r$ if $b = 0$ and $w = r + x \bmod (P-1)$ if $b = 1$.

4. Bob checks that $g^w = d_1 y^b$ and $h^w = d_2 h^b$

There are various differences between Protocols A and B. Protocol A is more efficient. Indeed with only one execution of the protocol, Alice has a $\frac{1}{P}$ probability of cheating. For protocol B Alice can cheat with probability $\frac{1}{2}$ and so the basic unit must be repeated many times in order to achieve a negligible cheating probability.

However in Protocol B the only messages sent by Bob (the Verifier) are his random coins. This makes Protocol B suitable to be made non–interactive by applying the Fiat–Shamir heuristic on top of it. Such option is not possible for Protocol A.

Finally, although here it is not an issue, Protocol B is perfect zero–knowledge, while Protocol A is only computationally zero–knowledge.

B ElGamal Fault-tolerant Threshold Cryptosystem

The protocols shown in Section 5 for MEG work also for the regular ElGamal cryptosystem.

Let us recall how ElGamal works.

p is a prime. g is a generator of Z_p^*. We assume that it is hard to compute discrete logs modulo p in Z_p^*. A random x in Z_{p-1} is the secret key and $y = g^x \bmod p$ the public key. To encrypt given a message v choose a random $r \bmod p-1$ and let $E_r(v) = (g^r \bmod p, y^r v \bmod p)$. To decrypt (c_1, c_2) let $v = \frac{c_2}{c_1^x} \bmod p$.

So it should appear clear how the scheme can be easily transported (including the fault-tolerance) to this case. The only sore point is the sharing of x that must be done modulo $p-1$ instead that modulo a prime q. This could be a problem because in order to perform Shamir's secret sharing protocol the secret must belong to a field. There are two solutions to this problem:

1. One could modify ElGamal's scheme to work with a prime p of the form $kq+1$ and g an element of order q. This is actually recommended, as it is believed that the discrete log problem is harder under these assumptions. In this case the sharing of x must be done modulo q and the construction of a fault-tolerant threshold cryptosystem for ElGamal is immediate.

2. If g is really a generator of Z_p^* then one can use the secret sharing techniques of De Santis et al [37] to share a secret over a ring.

Applications: Besides the application to election protocols outlined in this paper, fault-tolerant threshold cryptosystems have several other applications. Key-escrow is one of them. One could imagine the centers receiving shares of the key to be law enforcement agents and the decryption protocol $\mathcal{DP}$ being applied every time there is a need to decrypt a ciphertext. Notice that doing key-escrow this way has two main advantages:

1. *Absolute Warrant Bounds.* When a court-order is issued to decrypt a ciphertext, the secret key of the user is not compromised and so only that particular ciphertext is decrypted.

2. *Robustness.* Decryption is guaranteed even if up to a half of the centers refuse to participate or actively try to impede the process. This is particularly useful in scenarios in which the law enforcement agents do not trust each other (as in the international key escrow example in [21].)