Secure Voting Using Partially Compatible Homomorphisms

Kazue Sako¹ and Joe Kilian²

¹ NEC Corporation, 4-1-1 Miyazaki Miyamae, Kawasaki 216, JAPAN

² NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA

Abstract. We introduce a new number-theoretic based protocol for secure electronic voting. Our scheme is much more communication efficient than previous schemes of its type, and has a much lower round complexity than is currently possible using the anonymous-channel/mixer techniques. Preprocessing allows for nearly all of the communication and computation to be performed before any voting takes place. Unlike the mixer-based protocols, anyone can verify that everyone's vote has been properly counted. Also, our techniques allow for a wide variety of different schemes.

Our protocols are based on families of homomorphic encryptions which have a *partial compatibility* property, generalizing a method of Benaloh and Yung [2]. We use these functions to generate very simple interactive proofs on encrypted shares. We also develop amortization techniques yielding dramatic efficiency improvements over our simple protocols. Our protocols can be realized by current-generation PC's with access to an electronic bulletin board.

1 Introduction

Secure electronic voting is arguably the most important single application of secure multi-party computation. Yet despite extensive work on this subject, no complete solution has been found in either the theoretical or practical domains. Even the general solutions to secure multiparty protocols fail to exhibit all of the desired properties of elections (for example, the "receipt-free" property [1]). While these general solutions do have a wide breadth of security properties, and some hope of rigorous analysis, they are hopelessly impractical both in their computational and communication costs.

A number of more practical voting protocols have been proposed, with widely differing security properties. Schemes based on anonymous channels/mixers [5, 9, 12] have become very popular due to their superior efficiency and the arbitrary nature of the votes that are allowed. However, a price is paid for this efficiency. The simplest of these schemes cannot allow a voter to securely protest the omission of his vote without allowing a malicious voter to block the election. In all the schemes we know of there is a high round complexity - one round for each mixer used to implement the anonymous channel. Also, after the election each voter is typically responsible for checking that his vote was correctly tallied.

There is usually no way for an outside observer to later verify that the election was properly performed.

Another approach is the use of number theoretic techniques without anonymous channels or mixers [6, 3, 2]. The protocol has desirable security properties, but as we will discuss in detail, their communication complexity is quite high for realistic scenarios.

In this paper, we give a new number-theoretic scheme for secure electronic voting. This scheme has a number of features including moderate communication cost, low round complexity, preprocessing potential, security, universal verifiability and flexibility, which we outline below.

Communication Cost: Our protocol requires much less communication than the previous number-theoretic protocols. For one realistic setting of the parameters, we estimate that our protocol conservatively requires less than 1/20th the communication of Benaloh-Yung's scheme[2]. Furthermore, when more than one election is to be held, we can use an amortization technique that will boost our per-vote advantage to a factor of between 150 and 250. We note however that the communication complexity of each vote remains much greater than that required by the anonymous channel/mixer-based protocols. However, it is well within the range of feasibility, as we will discuss in greater detail in the Section 4.

Round Complexity: Our protocol enjoys an extremely low round complexity. Once the system has been set up, a voter can cast a vote in an election simply by posting a single message. The counting process consists of each counting center posting a single, very short message.

Preprocessing: Ideally, one would like to do the bulk of ones communication and computation in advance of an actual vote. In our scheme, one can preprocess ones vote with a single message. This preprocessing step doesn't depend on the vote that is eventually cast, and may be done at the time the voter "registers" to vote. When it comes time to actually vote, the voter can often simply post a single bit (or a bit along with a small integer in the worst case). Thus even with the signatures needed for identification, the communication cost is negligible. Similarly, after preprocessing, a PC acting as a voting center can easily count a hundred incoming votes per second.

Security: Under reasonable heuristic assumptions, no coalition of voters or centers can unfairly influence an election or significantly delay its outcome. A voter keeps his vote private as long as two of the centers he deals with are honest. The two honest center requirement can be reduced to a single honest center requirement by a simple doubling trick: each center simulates two centers of the original schemes.

While we use heuristic assumptions (such as the Fiat-Shamir method for noninteractive proofs) the only attack we know requires one to compute discrete-logarithms over the group we are working in. We can thus use elliptic curves for which the discrete-log problem is currently thought to be much harder than factoring. Previous number-theoretic approaches were based on the r th residue problem over Z_n^* , and are guaranteed breakable if one can factor n .

Universal Verifiability: Every action by a voter, whether preprocessing a vote or actually voting, is accompanied by a proof that the ballot is correctly constructed. The output of the counting center may also be easily checked for correctness. Any participant can verify that everyone's vote has been included in the tally. Once a party posts their message, it does not need to cooperate in the checking of its work. We can distribute the checking of the election over many parties, and if someone is still not satisfied they may check the results themselves. Thus, a voter has the option of minimally participating in an election by simply sending in their vote and then ceasing all involvement.

Flexibility: Our scheme is readily adaptable. For example, the number of centers can be made quite large. Voters may choose their own security/efficiency tradeoff and individually choose how many and which centers they use. Thus, we can practically support an election in which there are a 100 centers of which a typical voter chooses 10 and is protected as long as two of them are honest. For giant elections we can construct hierarchical counting schemes.

1.1 Techniques Used

New Techniques for Using Homomorphic Encryptions The main contribution of this paper is the use of a new tool: families of *partially compatible* homomorphic encryption functions. There are well known encryption functions with additive ($E(x + y) = E(x)E(y)$) or multiplicative $E(xy) = E(x)E(y)$ homomorphisms. These properties can be exploited to make very efficient zero-knowledge proofs, but can also work against security. For example, suppose that one has $E(x)$ and $E(y)$ and wishes to know whether $x + y$ is 1 or -1 . If E is a function (as opposed to a probabilistic encryption) with an additive homomorphism then one can compute $E(x + y)$ and check if it is equal to $E(1)$ or $E(-1)$.

Benaloh and Yung [2] circumvent this difficulty by using a family of probabilistic encryption functions, $\{E_1, \dots, E_n\}$. Each E_i probabilistically encrypts an element $x \in Z_r$, where r is a moderately large prime independent of i . While each E_i satisfies $E_i(x + y) = E_i(x)E_i(y)$, there is no obvious way of combining $E_i(x)$ and $E_j(y)$ to learn obtain an encryption of some simple function of x and y . A key requirement of their technique was that the encryptions E_i be probabilistic, a condition we weaken in the formalism below.

We consider a family of additive homomorphic, possibly deterministic encryption functions, $\{E_1, \dots, E_n\}$. Within this family we have a single group Z_q (where q is large) such that $E_i(x + y) = E_i(x)E_i(y)$, where $x, y \in Z_q$. Thus, they are all basically compatible. However, we require that for all (i, j) the following two distributions are computationally indistinguishable:

1. $(E_i(x), E_j(y))$, where x and y are chosen uniformly from Z_q , and
2. $(E_i(x), E_j(x))$, where x is chosen uniformly from Z_q .

This implies that for any v , $(E_i(x), E_j(v - x))$ is indistinguishable from $(E_i(x), E_j(y))$. Thus, if x and y are chosen uniformly such that $x + y = v$, then seeing $(E_i(x), E_j(y))$

doesn't reveal anything about v . Similarly, if $x_1, \dots, x_n$ are chosen uniformly to sum to v , then knowing $n - 2$ of the values $\{x_1, \dots, x_n\}$ tells one nothing about v .

We call such families of encryption functions with this property *partially compatible* homomorphic encryption functions. We don't know how to reduce the existence of such families to any well-known algebraic assumption. However, there are a number of candidates for such a family of encryption functions. For example, let primes $p_i = k_i q + 1$, where q is a prime, let g_i be a randomly (or pseudorandomly) chosen generator for $Z_{p_i}^*$ and let $\alpha_i = g_i^{k_i}$. If we define $E_i(x) = \alpha_i^x$ then we know of no way of obtaining any information about $x_1 + x_2$ given $E_1(x_1)$ and $E_2(x_2)$ without computing discrete logarithms. The only weakness we know of is that if $p_1 = p_2$ and ℓ such that $\alpha_2 = \alpha_1^\ell$ is known, then one can compute $E_1(x_1 + x_2)^\ell = E_1(x_1)^\ell E_2(x_2)$, which allows one to determine if $x_1 + x_2$ is equal to a given number. We know of no such attack when $p_1 \neq p_2$.

We can also incorporate encryption functions based on elliptic curves or other groups. Furthermore, we can mix which types of groups we use arbitrarily. For ease of exposition, we will use multiplicative notation for the cyclic group generated by α_i , regardless if the group is normally treated as multiplicative or additive.

Using these families of partially compatible homomorphic encryption functions, we construct very efficient interactive proofs for assertions such as:

- $x_1 + \dots + x_n = a + b$, given the encryptions for these values, and
- $x + y \in \{1, -1\}$, given the encryptions for x and y .

Because of the efficiency of these proofs, we can afford to run them many times, and use the heuristic trick of Fiat-Shamir [7] to make them noninteractive.

Reduced Costs Through Amortization Our improved proof techniques brings the complexity of the number-theoretic techniques down to the point where they can be used by a PC that can post messages to a bulletin board. However, when we use very strong confidence parameters (2^{-40} or even 2^{-60} error rates are recommended when using the Fiat-Shamir trick) and allow the voter to protect his vote by using many (e.g., 10) of the available centers, our costs are at the outer margin of usability. Hence, we have developed techniques for making these burdens easier to bear.

By allowing nearly all of the work to be done in advance of any election, we can amortize the computational and communication burden over a much larger period of time and still have very fast elections. To lower the computational burden of a proof, we use table lookup techniques to reduce the number of group operations required. Finally, we show how a voter can process many votes using much less communication and computation than would be required to process them individually.

We note that the use of amortization in cryptography is not new. Kurosawa and Tsujii [11] construct a zero-knowledge proof for two assertions that is more efficient than simply concatenating the zero-knowledge proofs for each assertion.

Boyar, Brassard and Peralta [4] and Kilian [10] consider the problem of achieving ultra-high confidence zero-knowledge proofs for NP using less communication than is required by simple sequential repetition. Franklin and Yung [8] show how to implement k instances of a secure multiparty computation much less expensively than k times the cost of a single secure computation.

1.2 Comparison to Related Work

Our work has been motivated by the work of Benaloh [6] and the later work of Benaloh and Yung [2]. The latter protocol enjoys most of desirable security properties we obtain, and is based on partially compatible homomorphisms of the form $E_i(x) = y_i^x \cdot g_i^r \bmod n_i$. The technical advances made by our work include:

1. Greater generality: The encryption used by [2] was tuned to the factoring problem. Each center i had the prime factorization of n_i as part of its secret information. This secret information complicated the protocol in that the voters needed to verify the correctness of the centers' public information and the correctness of their suballies through interactive protocols. Also, we can apply our method to "discrete-log type" problems - we don't know how to attack our scheme without computing discrete logarithms.
2. Amortization techniques: Unlike most previous work in voting, we consider how to run multiple elections more efficiently. Since there are usually many voters, and checking each vote involves many subchecks, we can effectively use amortization techniques to speed up single elections as well.
3. Improved zero-knowledge proofs: We give direct and efficient protocols to show, for example, that $x + y$ is either 1 or -1, without conveying which is the case. These proofs are more efficient than the *cryptographic capsule* (c.f. [3]) methods used in [6, 2].

Also, we incorporate techniques, such as the Fiat-Shamir heuristic for removing interaction, that were not available at the time of [2]. Some of these techniques can also be applied to the original [2] protocol (with varying degrees of difficulty and utility). By using more modern techniques we can better realize the basic approach laid out by Benaloh and Yung.

2 The Basic Scheme

We now describe the basic voting scheme. For simplicity, we assume that there are only two centers counting the votes, and that a single yes/no vote is being held. This scenario is solely to help understand the protocol given in Section 3. We note that this basic scheme does not protect privacy of one's vote against the centers. This issue will be overcome in Section 3, where we involve more than 2 centers.

2.1 Preliminaries and Initial Preparation

We denote the two centers by C_1 and C_2 . Each vote v will be broken into shares x_1 and x_2 , where x_i is a member of Z_q , and q is a prime. Before being posted, each share x_i is encrypted using encryption function E_i , where $\{E_1, E_2\}$ form a family of partially compatible homomorphic encryption functions.

As part of the setup process, which need only be done once for all time, the parties agree on $\{E_1, E_2\}$. Note that implementations based on discrete-log functions, there is no trapdoor information that need be kept hidden. Thus, for example a few bits from some global source can be fed into a pseudorandom bit generator and these random bits could be used to choose the moduli and generators needed to specify the desired functions. Heuristically, anyone can provide the seed to the pseudorandom generator, and it is unlikely that they will make it output a weak set of functions.

Along with setting up the family of encryption functions, we assume that basic primitives such as public-key cryptography and secure bit-string commitment have already been established. Let $H(x)$ denote a possibly probabilistic hash function that commits the sender to x without giving away any useful information about x .

2.2 The Election Procedure

The basic election runs in three stages: vote preparation, vote casting and vote counting.

Vote Preparation Each voter i chooses his vote v_i , 1 for yes-vote and -1 for no-vote. He uniformly generates $x_i^{(1)}$ and $x_i^{(2)}$ such that

$$x_i^{(1)} + x_i^{(2)} = v_i \bmod q.$$

He then posts $E_1(x_i^{(1)}) = \alpha_1^{x_i^{(1)}}$ and $E_2(x_i^{(2)}) = \alpha_2^{x_i^{(2)}}$ and proves $x_i^{(1)} + x_i^{(2)} \in \{1, -1\}$ without disclosing $x_i^{(1)}, x_i^{(2)}$ nor v_i . We discuss this proof in Section 2.3.

Voting Casting Each voter i encrypts $x_i^{(1)}$ and $x_i^{(2)}$ using the public keys of C_1 and C_2 respectively. Each center j computes $E_j(x_i^{(j)})$ and checks that it agrees with the previously posted value.

Counting Votes Each center j sums up $x_i^{(j)}$ modulo q for all voters i and posts sub-tally, t_j . Each voter verifies that

$$E_j(t_j) = \prod_i E_j(x_i^{(j)}),$$

and computes $T = t_1 + t_2$, which is equal to the number of “yes” votes minus the number of “no” votes.

2.3 Proving $x_1 + x_2 \in \{1, -1\}$

In Figure 1 we exhibit a simple protocol for proving that $x_1 + x_2 \in \{1, -1\} \bmod q$ given $E_1(x_1)$ and $E_2(x_2)$.

prove $\pm 1(x_1, x_2)$ /* Given $E_1(x_1), E_2(x_2)$, prove that
 $x_1 + x_2 \in \{1, -1\} \bmod q$ */

1. The prover uniformly chooses $r \in Z_q$ and $s \in \{1, -1\}$. He computes $R = H(r)$, the secure commitment for r , computes

$$Y_1 = E_1(s(x_1 + r)) = (E_1(x_1)E_1(r))^s \text{ and}$$

$$Y_2 = E_2(s(x_2 - r)) = (E_2(x_2)E_2(r)^{-1})^s,$$

and posts (Y_1, Y_2, R) .

- 2a. With probability $\frac{1}{2}$, the verifier asks the prover to reveal r and s . The verifier checks that r is consistent with R and that the above identities for Y_1 and Y_2 hold.
- 2b. With probability $\frac{1}{2}$, the verifier asks the prover to reveal $s(x_1 + r)$ and $t = s(x_1 + r) + s(x_2 - r) \in \{1, -1\}$. He then checks that $Y_1 = (E_1(x_1)E_1(r))^s$ and that $Y_2 = E_2(t - s(x_1 + r))$.

Fig. 1. Protocol for proving validity of shares.

Each execution of the above protocol will catch a cheating prover with probability $\frac{1}{2}$. We note that the distribution of (Y_1, Y_2) is easy to simulate given $(E_1(x_1), E_2(x_2))$. Indeed, if R is a perfect zero-knowledge bit commitment then the protocol is perfect zero-knowledge. We note that a conceptually more simple protocol would have the prover reveal $s(x_2 - r)$ in Step 2b. We chose the above protocol for its reduced communication complexity. Both s and t could also be eliminated by having the verifier check both possibilities, but this would save only 2 bits.

While we give this protocol in terms of a verifier, a more round efficient solution is to use the Fiat-Shamir method of eliminating interaction. First, the protocol is run many times (on the order of 40 or 60) in order to make the probability of withstanding all of the challenges vanishingly small. Then the verifier is replaced by a suitably "random looking" hash function which generates the challenges from the prover's posting in Step 1 of the protocol. If the prover is trying to prove an incorrect statement, then heuristically his only strategy is to run different postings through the hash function until he finds one whose challenges he can meet. However, the cost of this attack is prohibitive if the error probability is truly small (2^{-40} or 2^{-60}).

3 Many Centers and Many Votes

In the basic scheme we outline above, there were only two centers and a single yes-no vote. However, in more practical scenarios a voter will want to divide his vote among as many centers as possible - the more centers the more private the vote. Also, a voter is likely to participate in many elections and a given election is likely to have many yes/no votes. For example, Benaloh points out that approval voting (where a voter may cast a vote for any number of the given candidates) is really just a case of several independent yes/no votes[3]. In this section, we show how to split many votes over many centers with substantial amortized savings over preparing each vote separately.

For simplicity, we assume that there are only n centers and that each voter will split his votes over all n centers. For each center i we have an encryption function E_i from this family. Following the basic scheme, the voter breaks his vote $v \in \{1, -1\}$ into shares $X^{(1)}, \dots, X^{(n)}$ such that $v = X^{(1)} + \dots + X^{(n)}$, and then proves that these shares are correctly constructed. The most straightforward solution is to adapt protocol `prove±1` to handle more than two shares. Instead, we break the proof into two stages. First, the prover randomly generates a, b such that $v = a + b$ and proves that $X^{(1)} + \dots + X^{(n)} = A + B$. Then he uses `prove±1` to prove that $v = a + b$. This gives us chance to handle multiple votes efficiently as shown in Subsection 3.2.

3.1 Proving $X^{(1)} + X^{(2)} + \dots + X^{(n)} = A + B$

Figure 2 gives our protocol, `prove-sum`, for reducing a sum of n encrypted shares to a sum of 2 shares. We assume that the encryptions $E_i(X^{(i)})$, $E_a(A)$ and $E_b(B)$ are known, and that

$$(E_1, \dots, E_n, E_a, E_b)$$

is a family of partially compatible homomorphic encryptions with domain Z_q .

If the summation assertion is not true, then in each iteration of the protocol the prover will fail a check with probability at least $\frac{1}{2}$. As before, we lower this error rate to a very small value by repeated repetition, and then use the Fiat-Shamir heuristic to make the proof noninteractive.

3.2 Proving Equations Simultaneously

The bulk of computation and communication required for the full n -party scheme is taken up by the proof of the reduction to the 2-share sum. By combining many of these proofs into a single proof, the voter can efficiently prepare many “yes/no” ballots at once with significant savings in the amortized computation and communication required.

Suppose that the voter wants to prove that the following equations hold.

$$\begin{aligned} X_1^{(1)} + X_1^{(2)} + \dots + X_1^{(n)} &= A_1 + B_1 \\ X_2^{(1)} + X_2^{(2)} + \dots + X_2^{(n)} &= A_2 + B_2 \end{aligned}$$

prove-sum($X^{(1)}, \dots, X^{(n)}, A, B$)

/* Given $E_1(X^{(1)}), \dots, E_n(X^{(n)}), E_a(A), E_b(B)$, prove that $X^{(1)} + X^{(2)} + \dots + X^{(n)} = A + B$ */

1. For $0 \leq k \leq n$ the prover uniformly chooses $r_i \in Z_q$. He computes the commitment $R_i = H(r_i)$ and computes $Y_1, \dots, Y_n, Y_a, Y_b$ by

$$Y_i = E_i(X^{(i)} + r_i) \text{ for } 1 \leq i \leq n,$$

$$Y_a = E_a(A + r_0) \text{ and}$$

$$Y_b = E_b\left(B + \left(\sum_{i \geq 1} r_i\right) - r_0\right).$$

The prover posts $R_1, \dots, R_n$ and $Y_1, \dots, Y_n, Y_a, Y_b$.

- 2a. With probability $\frac{1}{2}$, the verifier challenges the prover to reveal $r_0, \dots, r_n$. The verifier checks that

$$R_i = H(r_i) \text{ for } 1 \leq i \leq n,$$

$$Y_i = E_i(X^{(i)})E_i(r_i) \text{ for } 1 \leq i \leq n,$$

$$Y_a = E_a(A)E_a(r_0), \text{ and}$$

$$Y_b = E_b(B)E_b\left(-r_0 + \sum_{i=1}^n r_i\right).$$

- 2b. With probability $\frac{1}{2}$ the verifier challenges the prover to reveal $(A + r_0), (B - r_0 + \sum_{i=1}^n r_i)$ and $(X^{(i)} + r_i)$ for $1 \leq i \leq n$. The verifier checks that

$$Y_i = E_i(X^{(i)} + r_i) \text{ for } 1 \leq i \leq n,$$

$$Y_a = E_a(A + r_0) = \alpha_a^{(A+r_0)} \text{ and}$$

$$Y_b = E_b\left(B + \left(\sum_{i \geq 1} r_i\right) - r_0\right) = \alpha_b^{(B + (\sum_{i=1}^n r_k) - r_0)}.$$

Finally, the verifier checks that

$$\sum_{i=1}^n (X^{(i)} + r_k) = (A + r_0) + \left(B - r_0 + \sum_{i=1}^n r_i\right).$$

Fig. 2. Protocol for proving summation assertions.

$$\vdots$$

$$X_m^{(1)} + X_m^{(2)} + \dots + X_m^{(n)} = A_m + B_m$$

and the values of $E_i(X_j^{(i)})$, $E_a(A_j)$ and $E_b(B_j)$ are known for $1 \leq i \leq n$ and $1 \leq j \leq m$. Let coefficients $c_1, \dots, c_m \in Z_q$ be chosen at random, and consider

the following linear equation:

$$X^{(1)} + X^{(2)} + \dots + X^{(n)} = \bar{A} + \bar{B}, \text{ where}$$

$$\begin{aligned} X^{(i)} &= \sum_{j=1}^m c_j X_j^{(i)} \\ \bar{A} &= \sum_{j=1}^m c_j A_j \text{ and} \\ \bar{B} &= \sum_{j=1}^m c_j B_j. \end{aligned}$$

By a simple probability argument, the following facts hold:

1. If all of the original linear equations were true, then the new linear equation will also be true, and
2. If at least one of the original linear equations is false, then the new linear equation will be false with probability $1 - 1/q$.

Thus, a proof of the new equation will suffice as a proof of *all* of the original equations.

It remains to show how to generate the encryptions for the new variables and how to choose the coefficients. The encryptions are given by

$$\begin{aligned} E_i(X^{(i)}) &= \prod_{j=1}^m E_i(X_j^{(i)})^{c_j} \\ E_a(\bar{A}) &= \prod_{j=1}^m E_a(A_j)^{c_j} \text{ and} \\ E_b(\bar{B}) &= \prod_{j=1}^m E_b(A_j)^{c_j}. \end{aligned}$$

We can view the c_i coefficients as challenges. As before, we can use the Fiat-Shamir trick and generate the c_i by a hash function of the original encryptions. Note that in this case, we do not have to perform the operation multiple times, since for a random setting of the coefficients an error in the original set of equations will result in an error in the final equation with all but vanishing probability. Indeed, for computational efficiency it suffices to choose c_i from $\{1, \dots, 2^{60}\}$, which will greatly speed up the exponentiations.

We give a summary of the election scheme in Figure 3.

m votes *n* centers Election Scheme

Precomputation (In the following description, *i* ranges over $1, \dots, n$, *j* ranges over $1, \dots, m$, and *k* ranges over the indices of all valid voters V_k .)

1. The vote counters, $C_1, C_2, \dots, C_n$ agree on a randomly selected family of partially compatible encryption functions, $E_a, E_b, \{E_i\}$ and post them.
2. Each voter V_k randomly chooses his "masking vote" $v_{k,j} \in \{1, -1\}$ and then randomly chooses two representations for $v_{k,j}$,

$$v_{k,j} = X_{k,j}^{(1)} + X_{k,j}^{(2)} + \dots + X_{k,j}^{(n)} = A_{k,j} + B_{k,j}.$$

He posts $E_i(X_{k,j}^{(i)})$, $E_a(A_{k,j})$ and $E_b(B_{k,j})$.

3. V_k executes the reduction given in Section 3.2 and obtains a single equation

$$X_k^{(1)} + X_k^{(2)} + \dots + X_k^{(n)} = \bar{A}_k + \bar{B}_k,$$

equality of which implies that the given two representations of masking vote $v_{k,j}$ are equal for all *j*. V_k then proves the validity of the equation using prove-sum.

4. V_k executes protocol prove±1 to show that $A_{k,j} + B_{k,j} \in \{1, -1\}$ for all *j*.
5. V_k encrypts $X_{k,j}^{(i)}$ using C_i 's public encryption algorithm, for all *j* and *i*, and posts the encryptions.

Vote Casting

1. To use his *j*th vote, voter *k* computes $s_{k,j} \in \{-1, 1\}$ such that his actual vote is equal to $s_{k,j}v_{k,j}$, and posts *j* and $s_{k,j}$. Note that often *j* is known and thus need not be sent. We assume that the correct *j* for each voter is known, and henceforth drop *j* from our notation.

Vote Counting

1. Each center C_i decrypts $X_k^{(i)}$ for all *k* and *j*, and verifies whether it is consistent with $E_i(X_k^{(i)})$. He calculates his subtotally $t^{(i)} = \sum_k s_k \cdot X_k^{(i)}$ and posts $t^{(i)}$.
2. Each person checking the vote verifies that $\prod_k (E_i(X_k^{(i)}))^{s_k}$ is equal to $E_i(t^{(i)})$. If so, they accept $\sum_i t^{(i)}$ as the tally of the vote.

If precomputation is not used, then v_k can be used as the actual vote and s_k is omitted.

Fig. 3. Overview of the general protocol.

4 Efficiency

4.1 Communication Cost

In this subsection, we roughly estimate communication cost of our scheme. While there are many possible variations of our scheme, a good understanding of their

complexity can be had by analyzing the cost of splitting a vote into encrypted shares and proving that the shares are well formed.

A number of security parameters come into play in this analysis. First, let us assume that our encryption functions are based on modular exponentiation over $Z_{p_i}^*$, and let k be an upper bound on the length of p_i (if different moduli are used, then they will not exactly be the same size). Let h be the output of the hash function H used for commitments and let l be the security parameter that effectively denotes how many times the proofs are run.

We consider the most general case of splitting m votes to n centers. Note that for m large, we achieve a higher amortized efficiency due to the method used in Section 3.2. Not counting the cost of the proof, representing these pieces along with the additional 2 shares used in our reduction requires $(n+2)km$ bits. The cost of proving the correctness of the combined equation (from Section 3.2) is $[2(n+2)k + (n+1)h]l$ bits. At this point, the voter has proved that each set of n shares representing a vote is equal to the two auxiliary shares. The proof that the two auxiliary shares sum to 1 or -1 costs $[3k + h]lm$ bits. The cost of revealing these shares to the proper counting authorities is approximately nk bits. Altogether, this gives a total of $2(n+1)mk + [(2n+2)k + (n+1)h + (3k+h)m]l$ bits. Some of the resulting numbers are shown in Table 1. If one uses the "center doubling" trick so as to need only one good center instead of two, then the costs are all doubled.

4.2 Computation Cost for Voters

In this subsection, we roughly estimate computation cost for the voters. In the scheme, the costly computations are mainly modular multiplication and modular exponentiation. Note that many modular exponentiations with the same base are being done. We can exploit this fact by computing lookup tables that will reduce the number of multiplications required by the exponentiations. For example, we can precompute α_j^i for all i 's that are powers of 2. This will reduce the average number of multiplications needed to compute $\alpha_x \bmod p$ from $\frac{3}{2}k$ to $\frac{1}{2}k$, requiring a table size of $(n+2)k^2$ bits. Using a more sophisticated table, we can obtain a further factor of 3 for the typical number ranges.

We again consider the case of splitting m votes into n shares each. Splitting m votes to mn pieces requires $\frac{1}{2}(n+2)km$ multiplications. $\frac{1}{2}(n+2)kl$ multiplications are needed for proving the reduction to the reduced 2-share representation of a vote. klm multiplications are needed to complete the proof that the votes are well formed. Verifying the suballies of each center requires $(\frac{1}{2}k + [\# \text{ of voters}])nm$ modular multiplications.

Altogether, our protocol requires approximately

$$\frac{1}{2}[(2n+2+2l)m + (n+2)l]k + [\# \text{ of voters}]nm$$

modular multiplications. A PC running at 33MHz can execute 768 multiplications in a second. Based on this, some of the resulting numbers are shown in Table 1.

Note that these figures are only approximate. However, the cost of the other modular addition or such operations as computing hash functions is comparatively negligible.

4.3 Computation Cost for Verifying Other Voters

In this subsection, we roughly estimate the computation cost needed for verification. Here again, k is the length of p , and l is a security parameter which determines maximum probability of cheating. c is the length of coefficients used in the protocol which can be set small. Also, modular exponentiation can exploit the previously mentioned table lookup techniques.

We again consider the case of splitting m votes into n shares each. $\frac{1}{2}(n+2)|c|(m-1)$ multiplications are needed for generating the encryption of the shares, including their representation. $\frac{1}{2}(n+2)(k+1)l$ multiplications are needed to verify that the combined equation is correct. $(k+1)lm$ multiplications are needed to complete the proofs that the shares are well formed. Altogether, this yields $\frac{1}{2}[(n+2)|c| + 2l(k-1))m + (n+2)(kl - l - |c|)]$ modular multiplications for each voter.

We can reduce this number by using techniques for verifying many modular exponentiations, resulting in a factor of 4 improvement over actually computing the exponentiations.

4.4 Efficiency Comparisons with Previous Work

The work of [2] gave the first protocol where votes are divided into pieces and the verifiable subtotally yields total outcome of voting. However, their scheme suffers from large communication complexity and seems not yet practical for implementation on existing networks. One of the reasons they need large communication complexity is that each centers i generate secret prime factors of their public key N_i . Therefore the scheme involves an interactive protocol to detect possible cheating at the setting of the public keys, together with an interactive protocol to show detected cheat was not due to a malicious voter. Also, since extra information of subtotally may reveal these secret primes, an interactive protocol was necessary to prove the correctness of subtotally. For above reasons, their protocol needed $(4l^2 + 5l + 2)kn$ bits for communication, where k is the size of the public keys of the n centers, and l is a security parameter.

The computation complexity is rather small for each iteration of their scheme, since the computation is based on $y^e x^r \bmod n$ where e and r are much smaller than n . However, since this interactive proof takes place many times, the total cost does not remain so small. We estimate their total computation assuming that they use the same trick constructing a table of $y^i \bmod n_j$ that requires nrk bits. Then, there will be $3(l^2 + 3l + 1)\lg rn + 2(l^2 + l + 1)n + [\# \text{ of voters}]n$ bit modular multiplication in total, where k is the size of public keys of the n centers, r determines number of voters and l is a security parameter. We give a rough numerical comparison in Table 1.

	1 vote, $l = 40$	1 vote, $l = 20$	100 votes, $l = 40$
Proposed Scheme	56K bytes 2.5 min.	28K bytes 1.5 min	1M bytes 58 min.
Benaloh & Yung (1000 voters)	4M bytes 11 min	1M bytes 3 min	400M bytes 19 hrs

All with $n = 10$ centers.

Table 1. Comparisons with the Benaloh - Yung Scheme

Acknowledgments

We would like to thank Mr. Masumoto of Central Labs for pointing out a vulnerability in our earlier scheme and for supplying us with data on the computational capabilities of various machines. We would further like to thank Josh Benaloh, Moti Yung and anonymous referees for their valuable comments.

References

1. Benaloh, J. C., Tuinstra, D.: Receipt-free secret-ballot elections. In STOC 94 (1994) pp. 544-553
2. Benaloh, J., Yung, M.: Distributing the power of a government to enhance the privacy of voters. In ACM Symposium on Principles of Distributed Computing (1986) pp. 52-62
3. Benaloh, J.: Verifiable secret-ballot Elections. PhD thesis Yale University 1987. YALEU/DCS/TR-561
4. Boyar, J., Brassard, G., Peralta, R.: Subquadratic zero-knowledge. In FOCS 91 (1991) pp. 69-78
5. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. In Communications of the ACM. ACM 1981 pp. 84-88
6. Cohen, J., Fischer, M. J.: A robust and verifiable cryptographically secure election scheme. In FOCS 85 (1985) pp. 372-382.
7. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In Advances in Cryptology -Crypto '86. Springer-Verlag 1986 pp. 186-199
8. Franklin, M., Yung, M.: Communication complexity of secure computation. In STOC 92 (1992) pp. 699-710
9. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In Advances in Cryptology -Auscrypt '92 (1992) pp. 244-251
10. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In STOC 92 (1992) pp. 722-732
11. Kurosawa, K., Tsujii, S.: Multi-language zero knowledge interactive proof systems. In Advances in Cryptology -Crypto '90 (1991) pp. 339-352
12. Park, C., Itoh, K., Kurosawa, K.: All/Nothing election scheme and anonymous channel. In Advances in Cryptology -EUROCRYPT '93 (1993)pp. 248-259