Hierarchical Distributed Election Protocols

Steve Chien Chung-chieh Shan

May 16, 1997

Abstract

In this paper, we survey and contribute to the field of secure distributed voting protocols. We begin by examining a series of increasingly sophisticated existing voting protocols, and demonstrate how they move from basic verifiability to add privacy and uncoercibility. We then propose a new protocol which allows a hierarchical voting scheme, which should be more scalable than existing protocols without practically sacrificing any of the other criteria.

Contents

Chapter 1

Introduction

The study of distributed voting protocols seeks to solve an easily stated problem, that of allowing networked voters to hold a secret-ballot election, presumably using cryptography. While many protocols have been proposed, each satisfying several desirable properties, algorithmic complexity was in general not a major consideration in the design of these protocols. Almost all distributed voting protocols proposed so far require a substantial amount of message passing and computation by a small number of authorities. As a result, these protocols are not particularly scalable.

In this paper, we explore the idea of hierarchical voting. Existing protocols use a flat voting structure, in which there is only one level of election authorities, and all voters report to all authorities. In our proposal, we allow for a hierarchical tree of voters. This reduces the number of messages required and lessens the computational load on the authorities and voters.

Depending on the requirements, it turns out that the addition of a hierarchical structure can either be done simply or with a fair amount of difficulty. If we think of the hierarchy as dividing a large election into smaller sub-elections whose tallies can be made public, then the problem is trivially solved. On the other hand, if we wish only to reveal the final overall tally, then a more complicated new protocol is needed. The new protocol must address several concerns, mostly having to do with verification of the intermediate authorities and maintaining voter privacy.

The organization of this paper is as follows. In Chapter 2, we will provide an overview of relevant research history and explore the cryptographic foundations which underlie distributed voting protocols. We will then describe these protocols in more detail and outline proofs of their correctness. In Chapter 3, we will present our new protocol, which we believe adds the scalability due to a hierarchical structure without sacrificing unduly any other desirable property. We will also consider how our solution can be applied to real-world network topologies, which in general do not provide the connectivity demanded by most distributed voting protocols. Finally, we will offer a comparison between our protocol and existing ones.

Chapter 2

Related Work

2.1 History and Overview

Research on the problem of cryptographic elections began in the mid-1980s. Since then, a fair amount of research has aimed to produce ever more sophisticated protocols with more and more desirable properties.

Most of the groundwork for distributed voting protocols came in the 1985 work of Cohen and Fischer [1], in which they first proposed a cryptographically secure election scheme. The Cohen/Fischer scheme uses a single election authority and satisfies three important properties:

- *Robustness:* Corrupted voters are unlikely to be able to prevent the election from completing successfully.
- *Verifiability:* A corrupt authority cannot fabricate the results of the election without being detected with high probability.
- *Security:* It is difficult to determine how any individual voter voted, unless the election authority is corrupt.

The Cohen/Fischer scheme has a number of problems, for which solutions have been subsequently proposed. The most obvious flaw is that the election is not truly secret, in the sense that the single election authority can decrypt and determine anyone's vote; the secrecy applies only to other voters. To solve this problem, Benaloh (Cohen) and Yung proposed in [2] to use multiple election authorities, all of which are required to cooperate in order to compute the final tally and decrypt any individual vote. Thus, only a government-wide conspiracy can compromise voter security.

Traditional paper ballot elections have still another advantage over the Benaloh/Yung proposal, that of uncoercibility. In a traditional election, once a voter has voted, there is no process by which the voter can prove how s/he voted. This property may not seem desirable at first, but this prevents the coercion of voters by heavily armed thugs who might otherwise demand proof of how a voter voted. The Benaloh/Yung scheme, like the Cohen/Fischer scheme and many other schemes, has the disadvantage of producing receipts that a voter can use to prove his/her vote. Benaloh and Tuinstra proposed a new protocol in [3] which avoids this problem to a certain extent.

2.2 Cryptographic Foundations

In this section, we explore the theoretical foundations which we use to build our distributed voting protocols.

2.2.1 Secret Sharing

One key part of all of our voting protocols are methods to allow for secure distributed computation. Here we outline three results which have proven useful in designing voting protocols. Please note that all computations below are carried out over a finite field F.

Shamir Secret Sharing

This classic 1979 result of Shamir allows a dealer D to distribute a secret value s to n players, such that at least $k \leq n$ players is required to reconstruct the secret. The protocol is *information theoretically secure*, i.e., any fewer than k players cannot gain any information about the secret by themselves. The protocol itself is very straightforward:

Protocol 1 (Shamir Secret Sharing) To share the secret s among players $P_1, P_2, \ldots, P_n$, such that k players are required to reconstruct the secret:

1. Dealer D creates a random polynomial $p(x)$ with degree $k-1$ and constant term s.
2. Dealer D publicly chooses n random distinct evaluation points $x_j \neq 0$, and secretly distributes to each player P_j the share $\text{share}_j(s) = p(x_j)$.

To reconstruct the secret from shares $\text{share}_1(s)$, $\text{share}_2(s)$, $\ldots$, $\text{share}_k(s)$:

1. Use Lagrange interpolation to find the unique polynomial $p(x)$ such that $\deg p(x) < k$ and $p(x_j) = \text{share}_j(s)$ for $j = 1, 2, \ldots, k$.
2. Reconstruct the secret to be $p(0)$. □

The key observation here is that k points are required to uniquely determine a polynomial with degree less than k. As a result, if only $k-1$ evaluation points are known, then for any constant s' we can create a unique and equally likely polynomial of degree $k-1$ by requiring the point $(0, s')$ to be on it. Thus, in our protocol k players can reconstruct the constant term s, but no smaller number can conspire to gain any information whatsoever. Reconstruction is done using Lagrange interpolation, a fairly standard technique. By performing $\binom{n}{k}$ reconstructions with the shares produced by each k-subset of players, Byzantine errors on the part of not too many players can be detected and corrected easily.

It is easy to extend Shamir secret sharing to let the players compute any linear combination of secrets without gaining information on intermediate results of the computation. To add two shared secrets together, the players need only add together individual shares at each evaluation point; to multiply a shared secret by a known scalar, just multiply each share by that scalar. It is more difficult, though still possible, to compute the product of two secrets, provided that at least half of the players follow the protocol correctly. Thus, it is possible to compute any "arithmetic" function (i.e., function involving only addition, subtraction, and multiplication) of secrets securely and robustly.

Sum Secret Sharing

Shamir's secret sharing protocol is flexible in that it allows the dealer to vary the number of players required to reconstruct the shared secret, by varying the degree of the polynomial $p(x)$. If we wish to require all players to cooperate in order to be able to reconstruct the secret, then we do not need a secret sharing protocol as (relatively) complicated as Shamir's. In this case, rather than resort to using a polynomial, we can simply distribute n shares whose sum is the desired secret.

Protocol 2 (Sum Secret Sharing) To share the secret s among players $P_1, P_2, \ldots, P_n$, such that all n players are required to reconstruct the secret:

1. Dealer D chooses n numbers $\text{share}_1(s), \text{share}_2(s), \ldots, \text{share}_n(s)$, such that the first $n-1$ are random, and $s = \sum_{j=1}^{n} \text{share}_j(s)$.

2. Dealer D distributes $\text{share}_j(s)$ to P_j for each j.

To reconstruct the secret from shares $\text{share}_1(s), \text{share}_2(s), \ldots, \text{share}_n(s)$, just compute $s = \sum_{j=1}^{n} \text{share}_j(s)$. □

Since each value for each individual share is equally likely, collecting together even $n-1$ of the shares affords no information about the secret. This protocol has the slight advantage over Shamir in that it is easier to compute.

Verifiable Secret Sharing

In terms of fault tolerance, both of the above protocols leave something to be desired. The sum secret sharing method is not fault-tolerant in the least, demanding that all participants cooperate fully. Shamir secret sharing detects and tolerates Byzantine faults in a certain number of players, but does not detect or tolerate errors on the part of the dealer, who may simply distribute random values as shares.

Fortunately, there exist secret sharing protocols which allow for secure distributed computations against Byzantine faults in both the dealer and the players. The protocols are rather complicated, and their internal mechanisms are not as important as their existence, so we will only present the result here. The most useful and sweeping result is that of T. Rabin and Ben-Or, who in 1989 constructed a protocol to compute any arithmetic function of secrets in the presence of at most $n/2$ Byzantine adversaries. As might be expected, these protocols are much more complex than Protocol 1.

2.2.2 Interactive Proofs

Frequently in our protocols, it is necessary for a participant to prove that it is behaving properly. For example, an election authority may need to certify that it has provided a valid public key for RSA encryption, or a voter may need to prove that it has actually voted a 0 (no) or a 1 (yes), rather than a 300. The difficulty is that often the prover cannot reveal too much information for fear of compromising the protocol; in our examples, the authority cannot demonstrate that the integer it has chosen is in fact the product of two large primes by revealing the primes, and the voter cannot decrypt the vote to reveal what it actually was. To overcome this problem, we will make frequent use of the concept of *interactive proofs*, pioneered by M. Rabin. Interactive proofs are a subclass of zero-knowledge proofs in which the prover is able to demonstrate probabilistically to a verifier's satisfaction that some property is true without revealing crucial information.

There are two broad categories of interactive proofs. The first approach can be used when there are many potential solutions and we need only find one that works. It does not matter which one, so long as the one we end up choosing is valid. Consider the case above, in which an authority wishes to prove that it has given an integer that is the product of two large primes, without revealing the prime factors themselves. In this case, as long as all we need is a valid such integer, we can ask the authority to produce n different candidate integers, each of which is valid. We then randomly choose one of them to actually use, and ask the authority to reveal the prime factors of the other $n-1$ integers. The authority will only be able to use a fraudulent key if the one we chose is the only corrupt one. Hence, the authority will remain undetected with probability $1/n$. If the punishment for attempting to use false keys is severe enough, this serves as a useful deterrent.

We can reduce the probability of non-detection significantly if there is related information we can use to prove knowledge without revealing critical information. The classic example often given here is that of graph isomorphism. Determining if two graphs are isomorphic is known to be an NP-complete problem, and we can imagine a situation in which a prover wishes to prove to a verifier that graphs A and B are isomorphic without revealing the actual isomorphism. The prover and verifier can carry out the following interactive proof:

Protocol 3 (Interactive Proof of Graph Isomorphism) To prove that graphs A and B are isomorphic with probability $1 - 1/2^n$:

1. Prover constructs a graph C that is isomorphic to A and B. This is easy to do since the prover knows the isomorphism; the prover need only randomly permute the vertices.

2. Verifier randomly chooses either A or B, and asks the prover to show how C is isomorphic to the selected graph.

3. Prover complies by demonstrating the proper mapping of vertices from C to the selected graph.

4. Repeat the above process n times. □

If the two graphs are not isomorphic or the prover does not know the isomorphism, then the prover's best strategy is to construct C isomorphic to either A or B, and hope that the verifier does not choose the other one. Thus, without knowing the isomorphism, the prover escapes each iteration with probability $1/2$. Over n iterations, the prover is successful with probability only $1/2^n$. Note also the crucial property: The verifier can gain no information no matter how many times the protocol is run, since for each graph C, it is shown only its isomorphism to one of A and B. This is a much stronger form of interactive proof, in which probability of fraud decays exponentially rather than inversely with respect to the number of iterations conducted, as was with the case above.

Interactive proofs will be used *ad nauseum* in the voting protocols presented below.

2.3 Existing Protocols

2.3.1 The Model

Before we move on to describe the voting protocols, we need to describe more formally the model we are using. The only common part among protocols in the literature is a known set of *voters,* each of whom must vote 0 or 1, with the final tally being the sum of the voters' votes. Most models also assume the existence of a *public communication channel,* via which authenticated messages can be reliably broadcast by all participants, such that all correct participants are guaranteed to receive the same set of messages. Many voting protocols, including ours, make use of other participants as *authorities,* responsible for collecting and counting votes. Several voting protocols also assume the existence of other devices, such as mechanisms for anonymous mail, or voting booths which can provide certain guarantees. One highly common device is that of a *beacon,* a publicly trusted source of broadcast random bits. It is generally possible to simulate this by having the participants generate random bits themselves and then XORing the bits together, but for convenience we will assume the beacon's existence when necessary.

We now define several properties we would like any voting protocol to have:

1. *Robustness:* Ideally, the presence of faulty participants (including misbehaving voters and corrupt authorities) should not hinder the successful completion of the election. If the election cannot be successfully completed, the portion of the election that must be restarted should be minimized. Less ideal but still acceptable would be to make certain that misbehaving participants can be detected; we can rationalize that a dire enough punishment for bad-faith failure will prevent corruption. Barely acceptable is to be able to detect that an election has been corrupted, without knowing who is responsible. This property is also known as *fault tolerance.*

2. *Verifiability:* Misbehaving authorities and voters should not be able to conspire to announce a false final tally. Ideally, any attempt at cheating will be detected with certainty, or very high probability, by honest participants.

3. *Privacy:* Any individual's vote should be private, in that no conspiracy of malevolent authorities or voters can determine any information about it beyond what they already know from the protocol. Often, the privacy ensured by voting protocols is not absolute,

but is based on the computational complexity of number theoretic problems. This property is also known as *security.*

4. *Uncoercibility:* A voter should not be able to prove to anybody else after the vote how it voted, and no adversary should be able to force the voter to disclose its vote even after corrupting a reasonable number of other participants. This property is also known as *receipt-free.*

5. *Scalability:* The message and computational complexity of the protocol should not increase dramatically with respect to the size of the election. Communication should be localized, and the amount of global broadcast minimized.

The series of existing protocols which we will summarize below will culminate in one which achieves all of the above except scalability. Our proposal will then add scalability to this list.

2.3.2 The Encryption Scheme

To start with, public-key encryption can serve the obvious purpose of turning any public communication channel into a private and/or authenticated one. In addition, most voting protocols below use the following scheme for encryption and decryption, which has an extremely useful homomorphism property that we will describe afterwards.

The encryption algorithm relies on three publicly known values n, y, and r. Of these, n must be the product of two large primes p and q, r should be a prime that divides either $p-1$ or $q-1$ but not both, and y should be relatively prime to n. To encrypt any value i mod r, we then pick some x relatively prime to n, and compute

$$w \equiv y^i x^r \pmod{n}. \tag{2.1}$$

It can be proven that there is only one value of i possible for each given w. In other words, each ciphertext w corresponds to only one plaintext value i. We write this as $w \in E(i)$, where

$$E(i) = \{\, w \mid \exists x,\, w \equiv y^i x^r \pmod{n} \,\}. \tag{2.2}$$

A term often used in the literature is that w is of *type* i. Note that, while each ciphertext only corresponds to one plaintext value, the converse is not true. Indeed, because x can vary, there are a great number of ciphertexts w available for each plaintext i. This probabilistic nature is a critical feature of this encryption scheme; it will ensure that encrypted votes cannot be compared for equality.

Another useful and easily verified property of this encryption scheme is that the product of two ciphertexts is an encryption of the sum of the corresponding plaintexts. In other words, if $w_1 \in E(i_1)$ and $w_2 \in E(i_2)$, then $w_1 w_2 \in E(i_1 + i_2)$. This property, called the *homomorphism property,* will allow us to compute the ciphertext of the tally from the ciphertext of individual votes, without having to decrypt each individual vote at any point in the process. We will also use this property heavily in interactive proofs described below.

For arbitrary i, the value x can be used as a *certificate*, or receipt, to prove that w is indeed a proper encryption of i. Without a certificate, decryption of a ciphertext w can be accomplished by exhaustively searching over all possible values of i mod r, and evaluating the expression

$$(wy^{-i})^{\phi(n)/r}, \tag{2.3}$$

which will be 1 if we have found the right value of i. (Here,

$$\phi(n) = (p-1)(q-1) \tag{2.4}$$

is the Euler ϕ function.) Note that only someone who knows $\phi(n)$ will be able to apply this technique, and that there is no known method to determine $\phi(n)$ without factoring n into p and q. In other words, assuming that p and q stays secret and that integer factorization stays difficult, only the entity that originally created the (n, y) pair will be able to computationally feasibly decrypt w using its knowledge of p and q.

One complicating factor remains. As noted above, to prove that w is an encryption of i, one can simply produce a certificate x, such that Equation 2.3 holds. Because x is randomly generated by the entity which encrypted i to w in the first place, it is information theoretically secure for this entity to release x mod n as a certificate to the public that $w \in E(i)$. However, consider the situation where w is a publicly known ciphertext, decrypted to i using the exhaustive search procedure described in the previous paragraph, by an entity that knows the factorization $n = pq$. Suppose that the entity wants to prove to the public that $w \in E(i)$; for example, one common case is that the entity is an election authority who wants to prove its decryption of the tally is correct. Although the exhaustive search procedure also produces a certificate x, this x is yet publicly unknown; therefore, releasing x may give an adversary additional information for factoring n. Hence, the entity needs to prove that $w \in E(i)$ without releasing x. To do this, the following sub-protocol may be used; it is a classical interactive proof protocol with exponentially decreasing probability of non-detection.

Protocol 4 (Interactive Proof of Decryption) Given n, r, y, w, i public as above, to prove with probability $1 - 1/2^N$ that Equation 2.3 holds for some x known privately by the prover:

1. The prover randomly and privately chooses N integers, $c_1, c_2, \ldots, c_N$. For each $j = 1, 2, \ldots, N$, the prover computes and releases C_j such that
$$C_j \equiv c_j^r y^{-i} \pmod{n}. \tag{2.5}$$
2. For each C_j, a random beacon bit b_j is obtained and one of the following steps performed:
   - If $b_j = 0$, the prover releases c_j, and the verifier checks that Equation 2.5 holds for this particular j. This proves that Equation 2.5 is true for all $j = 1, 2, \ldots, N$ with high probability.
   - If $b_j = 1$, the prover releases $c_j x$, and the verifier checks that
   $$C_j w \equiv (c_j x)^r \pmod{n}. \tag{2.6}$$

   This proves the above equation for all $j = 1, 2, \ldots, N$ with high probability. □

Note that, even though the verifier in this protocol can be convinced of the decryption to any probability arbitrarily close to 1, the verifier cannot, from executing the protocol, construct a certificate of the decryption. Nor can the verifier derive any information that would allow it to conduct a similar proof of decryption to anybody else.

A special case of the above protocol is particularly useful, namely when $i = 0$ and $w = w_1/w_2$ for some w_1, w_2. In this special case, the prover has proven that $w_1/w_2 \in E(0)$. By the homomorphism property mentioned above, the prover has proven that w_1 and w_2 correspond to the same plaintext, without revealing the plaintext itself. Of course, this can also be proven by revealing directly x such that $x^r \equiv w_1/w_2$, but as we explained above, this may be information theoretically insecure.

2.3.3 First Try: Cohen and Fischer, 1985

The Cohen and Fischer proposal of 1985 [1] provides robustness, verifiability, and some measure of privacy, and is built out of the tools discussed above. In this scheme, there is one election authority charged with computing and releasing the final tally, and all voters are able to verify for themselves that the final tally is correct. Voters vote by using *ballots,* or pairs of encrypted values (u, v), such that one value in the pair is an encryption of 0 and the other an encryption of 1. The voter votes by choosing either u or v to send in.

The scheme has four major steps:

1. The authority generates several sets of *encryption parameters,* i.e., values for n and y (r is assumed to be already publicly chosen to be much larger than the total number of voters), and interactively proves their validity, keeping one set of parameters for actual use.

2. Each voter prepares a set of ballots and interactively proves that they are valid, keeping one set of ballots for actual use.

3. Each voter votes by selecting half of the chosen ballot.

4. The authority counts the votes and releases the final tally.

In more detail, the protocol is as follows. All communications are assumed to be public.

Protocol 5 (Cohen/Fischer 1985 Election Scheme) In the following, the numbers k_1 and k_2 are security parameters that can be increased as deemed necessary to boost the probabilistic confidence in the election. Let Y denote the final tally.

1. Authority constructs parameters for encryption scheme.
   - Construct and release k_1 sets of valid encryption pairs (n_i, y_i).
   - Use the beacon to generate enough bits to randomly pick one of the encryption pairs.
   - For each encryption pair which was not chosen, prove that it is valid by releasing the two prime factors of n_i. The voters can then verify for themselves that the encryption pair satisfies all of the conditions listed in Section 2.3.2.

2. Voters prepare candidate ballots for use in the election.

   - Construct and release ballots (u_0, v_0), (u_1, v_1), $\ldots$, (u_{k_2}, v_{k_2}), such that for each i, either $u_i \in E(0)$ and $v_i \in E(1)$, or $u_i \in E(1)$ and $v_i \in E(0)$. Each voter plans to use (u_0, v_0) as the actual ballot, and now needs to prove its validity without revealing whether it is an encryption of $(0, 1)$ or $(1, 0)$.
   - Read k_2 bits from the beacon, $b_1, \ldots, b_{k_2}$.
   - For each $i = 1, \ldots, k_2$, perform one of the following steps:
     - If $b_i = 0$, provide certificates to show that either $u_i \in E(0)$ and $v_i \in E(1)$, or $u_i \in E(1)$ and $v_i \in E(0)$. This proves explicitly that the ith ballot is valid. (In this situation, it is secure to provide the certificates directly without going through Protocol 4 twice, because the x's used during encryption were randomly generated anyway.)
     - If $b_i = 1$, provide certificates to show that either u_i/u_0, $v_i/v_0 \in E(0)$ or u_i/v_0, $v_i/u_0 \in E(0)$. This demonstrates that the ith ballot and the actual ballot are encryptions of the same numbers, though not necessarily in the same order.

3. Voters perform the actual voting.

   - Select the desired half of (u_0, v_0), and submit it as the actual vote v.

4. Authority computes the final tally.

   - Compute the product of the encrypted votes the voters have submitted. The product will be an encryption of the sum of the unencrypted votes. In other words, compute the product over all voters $\prod v$, which will be an element of $E(Y)$, i.e., an encryption of the tally.
   - Decrypt $\prod v$ by exhaustively searching over all possible Y. Reveal Y, and invoke Protocol 4 to interactively prove that $\prod v \in E(Y)$ without producing an explicit certificate. □

Some of the above points may need further explanation:

In the interactive proof in step 1, the authority can only cheat if it guesses beforehand which set of encryption pairs the beacon bits will leave unchecked, and it can do this with probability $1/k_1$.

In the interactive proof in step 2, where each voter is proving that its ballots are valid, the idea is that for each test ballot (u_i, v_i), the voter will either have to reveal explicitly that it is valid, or reveal that it is of the same type as (u_0, v_0), the ballot to be actually used. If (u_0, v_0) is not a valid ballot, the voter will only escape undetected if it makes some of the test ballots valid and the rest of them of the same type as (u_0, v_0), and the beacon bits work out exactly properly. This requires that the voter guess beforehand each of the beacon bits, so the probability of escape is only $1/2^{k_2}$.

In step 4, since all communications are public, each voter can multiply all of the encrypted votes from step 3 for itself and verify using the authority's interactive proof that the final tally is indeed correct. If everything else has gone right, the authority can only escape with a false tally with probability 2^{-N}, where N is the security parameter chosen when invoking Protocol 4.

2.3.4 Improving Privacy: Benaloh and Yung, 1986

One obvious deficiency in the above protocol is that there is only one authority, which has the ability to decrypt any individual voter's vote. This hardly fits most definitions of a secret ballot, and thus Benaloh (the same person as Cohen above) and Yung proposed an improved scheme in 1986 [2]. In their new scheme, voters use sum secret sharing to distribute shares of their votes to each of T authorities, who then compute the sum from these shares. In this protocol, a vote now consists of a vector of encryptions of T integers; the plaintext should add to 0 or 1. More formally, a single voter's vote is now written as $(z_1, z_2, \ldots, z_T)$, where z_j is the jth authority's share of the vote. Since each voter only gets one vote, we require that $z_j \in E_j(m_j)$, where $\sum_j m_j = 0$ or 1. A valid ballot still consists of two votes, one of which sums up to 0 and the other of which sums up to 1. The key here is that each component of a vote is encrypted using a different authority's encryption parameters; because each authority t_j has its own encryption parameters (n_j, y_j), not revealed even to other authorities, all authorities must cooperate in order to decrypt any vote or tally. Note, however, that although n_j and y_j are different from authority to authority, r is still unique and global throughout the election.

The immediate problem that this adds is that it is now more difficult for the voters to prove that the ballots they use are valid, since the ballots now consist of many separate pieces. The protocol for verifying the honesty of the authorities is also more complicated this time.

The complete protocol is as follows; we explain parts of it in more detail afterwards.

Protocol 6 (Benaloh/Yung 1986 Election Scheme) Suppose there are T authorities, $t_1, \ldots, t_T$. We use E_j to denote the encryption function using the parameters (n_j, y_j) posted by authority t_j, and the global r value.

A. Verifying the honesty of the authorities.

1. Each authority t_j privately generates and provides encryption parameters (n_j, y_j), which it claims satisfies the necessary number theoretic properties.

2. Voters prepare k_1 test ballots and interactively prove each of them is valid. This uses a very similar protocol to that used in the interactive proof in step 2 of Protocol 5. For each test ballot (u_0, v_0) (where u_0 and v_0 are both vectors with T encrypted components), the voter carries out the following steps to prove the validity of the test ballot:

   - Prepare a certain number of auxiliary ballots (analogous to the (u_1, v_1), $\ldots$, (u_{k_2}, v_{k_2}) of Protocol 5.

   - For each auxiliary ballot (u_j, v_j), read a random bit from the beacon, and based on the beacon bit, perform one of the following (note that while this is the same as the above protocol in description, the mechanics are different given that each vote is now a vector and not an integer):

- If the beacon bit is 0, prove that (u_j, v_j) is valid by decrypting u_j and v_j. Each vote is decrypted by releasing certificates for each component of the vector, proving that the T plaintext components add up to 0 for one of the two votes and 1 for the other.
- If the beacon bit is 1, then prove that (u_j, v_j) is of the same type as (u_0, v_0), i.e., that either u_j is of the same type as u_0 and v_j as v_0, or u_j is of the same type as v_0 and v_j as u_0. (See description of this below.)

3. Each voter votes (randomly) one of the two halves of each of its test ballots; in effect, there are k_1 parallel elections being held.

4. The authorities decrypt all of the pieces of votes they receive and reveal the decryptions.

5. The voters release the plaintext of all of the pieces they voted, along with certificates, and a check is performed to make sure that all pieces in step 4 match those in step 5. If this is successful, then we can be sure with high probability that all of the authorities have distributed valid encryption parameters and are able to distinguish votes.

B. Conducting the actual election.

6. Each voter creates a new ballot for actual use, and uses the protocol in step 2 to verify that it is valid.

7. Each voter selects half of the new ballot and submits it as its actual vote. Voter j sends to authority i the share z_{ij} which is an encryption of m_{ij}.

8. Each authority now multiplies the shares it has received together to obtain $\prod_j z_{ij} \in E_i(\sum_j m_{ij})$. The authority now releases $\sum_j m_{ij}$ and invokes Protocol 4 to prove that it is the decryption of $\prod_j z_{ij}$.

9. The final tally for the election is simply the sum of the released by the authorities (mod r). □

This protocol is structurally the same as Protocol 5. The only complicating factor is in step 2, where we need to prove that two votes are of the same type. This can be accomplished using a method similar to the dividing of encrypted votes in Protocol 5, where the quotient of the encrypted votes was shown to be an encryption of 0. Here, with votes being vectors instead of integers, we need to construct a vector of quotients and show that the quotients are encryptions of integers which add up to 0. More formally, let one vote be $(a_1, \ldots, a_T) \in \{u_j, v_j\}$ and the other be $(a'_1, \ldots, a'_T) \in \{u_0, v_0\}$, where $a_j \in E_j(m_j)$, $a'_j \in E_j(m'_j)$. If the two votes are indeed of the same type, m_j and m'_j would satisfy

$$\sum_{j=1}^{T}(m_j - m'_j) \equiv 0 \pmod{r}. \tag{2.7}$$

For each j, let x_j and x'_j be certificates for $a_j \in E_j(m_j)$ and $a'_j \in E_j(m'_j)$, respectively; in other words, let x_j and x'_j be the random numbers used when the voter prepared the votes in the first place, such that

$$a_j \equiv y_j^{m_j} x_j^r \pmod{n_j}, \tag{2.8}$$

$$a'_j \equiv y_j^{m'_j} x_j'^r \pmod{n_j}. \tag{2.9}$$

Then, we have

$$a_j/a'_j \equiv y_j^{m_j - m'_j} \left(\frac{x_j}{x'_j}\right)^r \pmod{n_j}, \tag{2.10}$$

$$a_j/a'_j \equiv y_j^{m_j - m'_j} y_j^r \left(\frac{x_j}{x'_j y_j}\right)^r \pmod{n_j}. \tag{2.11}$$

If $m_j - m'_j \geq 0$, the voter releases $m_j - m'_j$ and certifies it with x_j/x'_j. If $m_j - m'_j < 0$ on the other hand, the voter releases $m_j - m'_j + r$ and certifies it with $x_j/x'_j y_j$. This normalization ensures that the decryption released for a_j/a'_j always falls within the range $[0, r-1]$, so that no useful information to the adversary is revealed about the relative order between m_j and m'_j.

This completes our description of Protocol 6. By tuning the security parameters including k_1 and k_2, any level of confidence in the results of the election can be achieved. It should be noted that, consistent with the principles of sum secret sharing, all of the authorities are required to collaborate in order to determine any individual's vote. The down side of this, however, is that a single authority's crash failure can halt the election.

2.3.5 Improving Uncoercibility: Benaloh and Tuinstra, 1994

Benaloh continued his crusade for better election protocols with his 1994 joint effort with Tuinstra [3], in which they propose a solution to another flaw in Protocols 5 and 6.

In the protocols above, all of the voters' communications are public, and thus the voters are subject to coercion to reveal their votes. An armed villain could conceivably collect all of the shares a voter sent during the election and demand under extreme duress that the voter reconstruct his or her actual vote. What is missing is some private information that the voter can use to lie about his or her vote. In [3], Benaloh and Tuinstra fix this and thus make their new protocol uncoercible under certain circumstances.

The model used in this protocol is very similar to that of the two previous protocols, with multiple voters and multiple authorities. To allow for the communication of private information, Benaloh and Tuinstra assume the existence of a private channel between each authority and voter. The new proposal also changes the fault tolerance requirement from Protocol 6, by allowing for some authorities to fail. This is accomplished by using Shamir secret sharing rather than sum secret sharing.

The key point in this protocol is that each authority transmits a private, random masking value to each voter "inside the voting booth." The voter then knows to adjust the component in its vote corresponding to that authority by the masking value. After exiting the voting booth, the voter is able to lie about the masking value, thus frustrating the attempts of various armed henchpersons. At the end of the protocol, the total of the masking values for each voter is removed from the decrypted tally, resulting in the final, correct tally. While the masking values themselves are transmitted privately, encryptions of the masking values are released to the public, thus ensuring that the authority cannot falsify the result of the election by lying about the masking values.

Each vote is now shared among authorities based on points on a polynomial, with polynomials having constant term 0 representing a 0-vote and 1 representing a 1-vote. Each voter will send to each authority its share of the polynomial translated by the private masking value, along with the encrypted masking value. Thus, given a secret vote $s \in \{0, 1\}$ for which the voter has chosen the shares $\mathrm{share}_1(s)$, $\mathrm{share}_2(s)$, $\ldots$, $\mathrm{share}_T(s)$ with Shamir secret sharing, the masked share of s sent to authority j is $\mathrm{share}_j(s) + x_j$ rather than $\mathrm{share}_j(s)$, where x_j is the private masking value transmitted by authority t_j to the voter.

The complete protocol is as follows:

Protocol 7 (Benaloh/Tuinstra 1994 Election Scheme) Again, suppose there are T authorities, $t_1, \ldots, t_T$. We use E_i^* to denote the encryption function using the parameters (n_j^*, y_j^*) posted by authority t_j, and the global r value. (We use the asterisks to emphasize the fact that encryption is only used for the authorities to publicly commit to their masking values. No encryption is needed for the actual votes, because they are already masked anyway.) Let Y denote the final tally.

1. Authorities each construct and prove the validity of encryption parameters (n_j^*, y_j^*), using one of the previously described protocols.

2. For each voter, each authority t_j randomly selects a masking value x_j and other values $c_1, \ldots, c_{k_1}$. The plaintext values x_j, $c_1, \ldots, c_{k_1}$ are transmitted privately to the voter in the voting booth, while the encrypted values $z_j^* \in E_j^*(x_j)$, $s_1 \in E_j^*(c_1)$, $\ldots$, $s_{k_1} \in E_j^*(c_{k_1})$ are publicly broadcast.

3. In order for the authority to interactively prove that the plaintext values correspond to the encrypted values, the beacon then generates k_1 random bits. For $i = 1, \ldots, k_1$, one of the following is done:

    - If the ith beacon bit is 0, the authority decrypts s_i and shows that it is an encryption of c_i by presenting a certificate.
    - If the ith beacon bit is 1, the authority decrypts $s_i z_j^*$ and shows that it is an encryption of $c_i + x_j$ by presenting a certificate.

4. Each voter now creates $k_2 + 1$ ballots, $B_0, B_1, \ldots, B_{k_2}$, each comprised of a 0-vote and a 1-vote, and publicly distributes to each authority its masked share of both halves of the ballot, each component masked using different masking values. There are $T(k_2+1)$ masking values involved in total for each voter; denote the masking value for ballot B_i and authority t_j as x_{ij}, an encryption $z_{ij}^* \in E_j^*(x_{ij})$ of which is publicly released ($0 \le i \le k_2$, $1 \le j \le T$). With the use of k_2 beacon bits, the following familiar interactive proof is then executed for each ballot $B_1, \ldots, B_{k_2}$, which will show with high probability that B_0 is a valid ballot:

- If the ith beacon bit is 0, each authority t_j decrypts its masking value x_{ij}. Everyone can then subtract x_{ij} from the masked shares to obtain unmasked shares, and reconstruct the secret vote pair to verify that one vote is 0 and the other is 1.
- If the ith beacon bit is 1, the voter shows that the two halves of B_0 correspond to the two halves of B_i by instructing the authorities as to which half of one corresponds to which half of the other. For each pair of votes

$$\begin{aligned} (\mathrm{share}_1(s) + x_{01}, \mathrm{share}_2(s) + x_{02}, \ldots, \mathrm{share}_T(s) + x_{0T}) \quad &\text{in} \quad B_0, \\ (\mathrm{share}_1(s') + x_{i1}, \mathrm{share}_2(s') + x_{i2}, \ldots, \mathrm{share}_T(s') + x_{iT}) \quad &\text{in} \quad B_i \end{aligned}$$

which the voter claims to be of the same type (i.e., $s = s'$), each authority t_j reveals $x_{0j} - x_{ij}$ and certifies that $z^*_{0j}/z^*_{ij} \in E^*_j(x_{0j} - x_{ij})$ without revealing x_{0j} or x_{ij}. Everybody can then compute

$$\begin{aligned} &(\mathrm{share}_j(s) + x_{0j}) - (\mathrm{share}_j(s') + x_{ij}) - (x_{0j} - x_{ij}) \\ &\quad = \mathrm{share}_j(s) - \mathrm{share}_j(s') = \mathrm{share}_j(s - s') \end{aligned} \tag{2.12}$$

for each j, thereby reconstructing $s - s'$, which should be zero (mod r).

5. Each voter selects half of B_0 and submits it as its actual vote. To vote $s \in \{0, 1\}$, the voter selects the half

$$(\mathrm{share}_1(s) + x_{01}, \mathrm{share}_2(s) + x_{02}, \ldots, \mathrm{share}_T(s) + x_{0T}) \tag{2.13}$$

from B_0.

6. Everybody can now calculate the sums over all voters

$$\begin{aligned} &\Big(\sum(\mathrm{share}_1(s) + x_{01}), \sum(\mathrm{share}_2(s) + x_{02}), \ldots, \sum(\mathrm{share}_T(s) + x_{0T})\Big) \\ &\quad = \Big(\mathrm{share}_1(Y) + \sum x_{01}, \mathrm{share}_2(Y) + \sum x_{02}, \mathrm{share}_T(Y) + \sum x_{0T}\Big) \end{aligned} \tag{2.14}$$

where Y is the correct tally. Everybody can also calculate the products over all voters

$$\Big(\prod z^*_{01}, \prod z^*_{02}, \ldots, \prod z^*_{0T}\Big) \tag{2.15}$$

in which $\prod z^*_{01} \in E^*_1(\sum x_{01})$, $\prod z^*_{02} \in E^*_2(\sum x_{02})$, $\ldots$, $\prod z^*_{0T} \in E^*_T(\sum x_{0T})$. Each authority t_j now decrypts $\prod z^*_{0j}$, revealing $\sum x_{0j}$ and certifying that $\prod z^*_{0j} \in E^*_j(\sum x_{0j})$. Everybody can now calculate

$$\begin{aligned} &\Big(\mathrm{share}_1(Y) + \sum x_{01}, \mathrm{share}_2(Y) + \sum x_{02}, \mathrm{share}_T(Y) + \sum x_{0T}\Big) \\ &\quad - \Big(\sum x_{01}, \sum x_{02}, \sum x_{0T}\Big) = (\mathrm{share}_1(Y), \mathrm{share}_2(Y), \mathrm{share}_T(Y)). \end{aligned} \tag{2.16}$$

From these unmasked shares of Y, everybody can then reconstruct the tally Y. □

The key part of this protocol is that the correctness of the masking values is proven without resorting to certificates; thus, a voter has no way of proving what its masking values were, and can thus lie about them. If several of the authorities are corrupt and reveal their masking values, a voter can still lie by being honest about the shares sent to corrupt authorities and deceptive about the shares sent to honest authorities. As long as at least one authority is trustworthy, the voter can lie about that authority. Of course, this means that a voter has to know beforehand which authorities are trustworthy, and which is problematic. Still, given that assumption, this protocol achieves uncoercibility.

In later work done in 1996 [4], Wu created a new protocol with slightly different assumptions, in which it is not necessary to know beforehand which voters are corrupt. Wu's protocol relies heavily on verifiable secret sharing, but is clean and elegant. In summary, Wu introduces the idea of a *polling device,* which is able to transmit bits to a voter anonymously. Each authority uses VSS to share a bit among the other authorities. The set of bits generated this way is communicated to the voter privately and anonymously, and the voter uses the XOR of these bits to mask the shares it sends out. The authorities then remove the mask and reconstruct the vote. All the voter needs to do to lie about its vote is lie about one of the bits it receives.

Chapter 3

Hierarchical elections

We are now ready to present original results on making election protocols scalable by distributing the election across a wide-area network. All current election protocols require some kind of global state to be maintained, for example the "bulletin board" for public communication, which subsumes the bit per eligible voter indicating whether the voter has voted or not. Hence, it is desirable for as much global state as possible to be localized in the network topology. A relatively straightforward way to make distribution more scalable is to make the election hierarchical. To this end, we propose the following model:

1. The election is divided into D *districts*. Each district has its own disjoint set of voters. Votes from each district are counted separately, then merged into a global tally.

2. Each district has a number of *intermediate authorities*, whose role is similar to the authorities in the existing protocols described above. Voters from each district interact directly only with intermediate authorities. Intermediate authorities are responsible for passing per-district tallies up to higher-level authorities. For simplicity, we assume that the number of intermediate authorities in each district are the same (namely T), but this need not be true.

3. In addition to the intermediate authorities, there are U *global authorities*, responsible for receiving district tallies from intermediate authorities and computing the global tallies.

The assumption in this model is that voters in each district care enough about the correctness of their own district-wide election, that any violation of the protocol will always be detected by some voter within the same district. Hence, much of the information that was globally broadcast in previous election protocols need only be broadcast at the district level in order for the election to be verifiable.

In this model and probably any hierarchical election protocol, there is a fundamental tradeoff for efficiency and robustness at a slight expense of voter privacy. The reason is that, compared to a non-hierarchical election—a special case of the above model, in which there is only one district with DT "intermediate" authorities—the number of authorities contacted by each voter is now smaller. A smaller number of authorities implies reduced message and computation complexity, but the maximum number of authorities that can be corrupt before voter privacy is compromised is also forced to be smaller. For example, with sum secret sharing, the number of authorities that need to be corrupt in order to violate privacy is exactly the number of authorities that are contacted by the voter. With other secret sharing schemes such as Shamir secret sharing, fewer authorities may need to be corrupt, but never more.

3.1 Public district tallies

If it is deemed acceptable for per-district tallies to be made public, running a hierarchical election becomes a simple matter of running several parallel district-wide elections. Any existing election protocol with the desired properties can be used for the district-wide elections, resulting in public district tallies that can then be added up by the general public. No global authorities are needed in this scheme.

If sum secret sharing (as opposed to Shamir secret sharing) is used, and it is desired to further reduce communication complexity in this scheme, a tradeoff reducing both complexity and potential privacy can be made as follows. Instead of every voter contacting all T intermediate authorities in the district, the voter can contact only $S < T$ authorities. As before, the voter still engages in an interactive proof to show that the ballot is valid; however, for all intermediate authorities not explicitly contacted by the voter, the component type for that authority defaults to 0 mod r (i.e., the encrypted component defaults to 1 mod n_j for authority t_j). While obviously reducing the amount of communication in the protocol, this change also reduces the maximum number of corrupt authorities needed to violate a voter's privacy from T to S. Depending on whether S is a large number itself, this may or may not be a sensible tradeoff to make. We term this tradeoff *subset authority voting*. Unfortunately, subset authority voting only works for sum secret sharing, not Shamir secret sharing.

3.2 Private district tallies

A more interesting requirement for a hierarchical election scheme is that the district tallies remain secret. To satisfy this requirement, we first modify Protocol 7 to produce *encrypted* masked shares of the district tally, rather than unencrypted masked shares. This can be achieved by encrypting the votes sent from the voter to the authorities.

In this scheme, to submit a secret vote $s \in \{0, 1\}$ for which the voter has chosen the shares $\mathrm{share}_j(s)$ with Shamir secret sharing, the voter no longer sends $\mathrm{share}_j(s) + x_j$ publicly to authority t_j (where x_j is the private masking value transmitted by authority t_j to the voter), as in Protocol 7. Instead, the voter sends an encryption $a_j(s) \in E_j(\mathrm{share}_j(s) + x_j)$, which can be decrypted only by t_j. As we will see below, this ensures that district tallies remain secret. Note that the encryption parameters used by voters to encrypt the votes are different from the parameters used by authorities to publicly commit to the masking values; if the same set of parameters were used, uncoercibility would be lost. We detail the modifications below.

Protocol 8 (District Election Scheme) The notation is same as that in Protocol 7, except we use E_j in addition to E_j^* to denote the encryption function using the parameters

(n_j, y_j) posted by authority t_j. The global r value is common between the two sets of parameters (and across all authorities, as usual).

1. Authorities each construct and prove the validity of encryption parameters (n_j, y_j) and (n_j^*, y_j^*), using one of the previously described protocols.

2. For each voter, each authority t_j randomly selects a masking value x_j and other values $c_1, \ldots, c_{k_1}$. The plaintext values $x_j, c_1, \ldots, c_{k_1}$ are transmitted privately to the voter in the voting booth, while the encrypted values $z_j^* \in E_j^*(x_j)$, $s_1 \in E_j^*(c_1), \ldots, s_{k_1} \in E_j^*(c_{k_1})$ are publicly broadcast. Just as in step 3 of Protocol 7, the authority interactively proves that the plaintext values actually correspond to the encrypted values.

3. Each voter now creates k_2+1 ballots, $B_0, B_1, \ldots, B_{k_2}$, each comprised of a 0-vote and a 1-vote. Again, note that each vote now consists of T shares which are *encrypted* (using E_j) and masked. Each voter publicly distributes to each authority its encrypted masked share of both halves of the ballot, each component masked using different masking values. There are $T(k_2+1)$ masking values involved in total for each voter; denote the masking value for ballot B_i and authority t_j as x_{ij}, an encryption $z_{ij}^* \in E_j^*(x_{ij})$ of which is publicly released ($0 \le i \le k_2$, $1 \le j \le T$). With the use of k_2 beacon bits, the following interactive proof is then executed for each ballot $B_1, \ldots, B_{k_2}$, which will show with high probability that B_0 is a valid ballot:

   - If the ith beacon bit is 0, each authority t_j decrypts its masking value x_{ij}, and the voter decrypts the encrypted masked share that it previously sent to each authority. The authorities certify their decryption of the masking values using a certificate with respect to E_j^*; the voter certify its decryption of the ballot shares using a certificate with respect to E_j. Everyone can then subtract x_{ij} from the decrypted masked shares to obtain unmasked shares, and reconstruct the secret vote pair to verify that one vote is 0 and the other is 1.
   - If the ith beacon bit is 1, the voter shows that the two halves of B_0 correspond to the two halves of B_i by instructing the authorities as to which half of one corresponds to which half of the other. For each pair of votes

     $$\begin{aligned} (a_1(s), a_2(s), \ldots, a_T(s)) &\quad \text{in} \quad B_0, \\ (a_1(s'), a_2(s'), \ldots, a_T(s')) &\quad \text{in} \quad B_i \end{aligned}$$

     (where

     $$\begin{aligned} a_j(s) &\in E_j(\text{share}_j(s) + x_{0j}), \\ a_j(s') &\in E_j(\text{share}_j(s') + x_{ij}), \end{aligned}$$

     for each j) which the voter claims to be of the same type (i.e., $s = s'$), each authority t_j reveals $x_{0j} - x_{ij}$ and certifies that $z_{0j}^*/z_{ij}^* \in E_j^*(x_{0j} - x_{ij})$, without revealing x_{0j} or x_{ij}. The voter, on the other hand, reveals

     $$(\text{share}_j(s) + x_{0j}) - (\text{share}_j(s') + x_{ij}) \tag{3.1}$$

and certifies that

$$\frac{a_j(s)}{a_j(s')} \in E_j((\mathrm{share}_j(s) + x_{0j}) - (\mathrm{share}_j(s') + x_{ij})), \tag{3.2}$$

without revealing $\mathrm{share}_j(s)+x_{0j}$ or $\mathrm{share}_j(s')+x_{ij}$. Everybody can then compute

$$\begin{aligned}(\mathrm{share}_j(s) + x_{0j}) - (\mathrm{share}_j(s') + x_{ij}) - (x_{0j} - x_{ij}) \\ = \mathrm{share}_j(s) - \mathrm{share}_j(s') = \mathrm{share}_j(s - s')\end{aligned} \tag{3.3}$$

for each j, thereby reconstructing $s - s'$, which should be zero (mod r).

4. Each voter selects half of B_0 and submits it as its actual vote. To vote $s \in \{0, 1\}$, the voter selects the half

$$(a_1(s), a_2(s), \ldots, a_T(s)) \tag{3.4}$$

from B_0.

5. Everybody can now calculate the products over all voters

$$\left(\prod a_1(s), \prod a_2(s), \ldots, \prod a_T(s)\right). \tag{3.5}$$

Everybody can also calculate the products over all voters

$$\left(\prod z^*_{01}, \prod z^*_{02}, \ldots, \prod z^*_{0T}\right) \tag{3.6}$$

in which

$$\prod z^*_{0j} \in E^*_j\left(\sum x_{0j}\right) \tag{3.7}$$

for each j. Each authority t_j now decrypts $\prod z^*_{0j}$, revealing $\sum x_{0j}$ and certifying that $\prod z^*_{0j} \in E^*_j(\sum x_{0j})$. Each authority (or somebody other random entity) then encrypts and certifies $\sum x_{0j}$ into $z_{0j} \in E_j(\sum x_{0j})$ (note the transition between encryption parameters, from E^*_j to E_j, at this stage). Because for all voters

$$a_j(s) \in E_j(\mathrm{share}_j(s) + x_{0j}) \tag{3.8}$$

and the homomorphism properties of E_j and share, we have

$$\prod a_j(s) \in E_j\left(\mathrm{share}_j(Y) + \sum x_{0j}\right) \tag{3.9}$$

for each j, where Y is the correct tally. Hence, again by the homomorphism property of E_j, everybody can calculate

$$Z_j = \frac{\prod a_j(s)}{\prod z_{0j}} \in E_j\left(\mathrm{share}_j(Y)\right). \tag{3.10}$$

6. The general public now knows with high probability the encrypted but unmasked shares
$$(Z_1, Z_2, \ldots, Z_T) \tag{3.11}$$
of Y. If we wish to know the actual tally Y, we can simply ask each authority t_j to decrypt (by exhaustive search) $\text{share}_j(Y)$ and execute Protocol 4 to prove the correctness of the decryption. Everybody then knows the unencrypted, unmasked shares
$$(\text{share}_1(Y), \text{share}_2(Y), \ldots, \text{share}_T(Y)), \tag{3.12}$$
from which everybody can reconstruct the tally Y. However, for use as a sub-protocol in the hierarchical election protocol below, this step must not be executed. The privacy of the district tally is preserved in the same way in which the privacy of each individual voter's vote is preserved, namely Shamir secret sharing (or any other secret sharing scheme that is plugged in). □

Based on Protocol 8 for running each district election, we can then run a hierarchical election using the following protocol.

Protocol 9 (Hierarchical Election Scheme) Denote the intermediate authorities in the election by $t_1, \ldots,$ and the global authorities by $t'_1, \ldots, t'_U$. Denote by $Y_1, \ldots, Y_D$ the district tallies, and $Y = \sum_{d=1}^{D} Y_d$ the global tally. We use E_i to denote the encryption function using the parameters (n_j, y_j) posted by intermediate authority t_i, and the global r value. We use E'_i to denote the encryption function using the parameters (n'_j, y'_j) posted by global authority t'_i, and the global r value. Note that, unlike with public district tallies, r here must be chosen to be larger than the total number of voters in the global election.

1. For each district, run a district sub-election using Protocol 8. However, stop before step 6. At this point, each authority t_j has a secret share $\text{share}_j(Y_d)$ of the district tally Y_d, and encryptions
$$Z_j \in E_j(\text{share}_j(Y_d)) \tag{3.13}$$
are publicly known. (Here d denotes the district to which t_j belongs.)

2. Global authorities each construct and prove the validity of encryption parameters (n'_j, y'_j), using one of the previously described protocols.

3. Each intermediate authority t_j privately creates shares $\text{share}'_i(\text{share}_j(Y_d))$ for $i = 1, \ldots, U$ using a secret sharing scheme, and (publicly) sends
$$A_{ij} \in E'_i(\text{share}'_i(\text{share}_j(Y_d))) \tag{3.14}$$
to the global authority t'_i.

4. Each t_j also privately picks K random numbers $c_1, c_2, \ldots, c_{k3}$ and releases encryptions $C_k \in E_j(c_k)$ for $k = 1, 2, \ldots, K$. For each k, t_j privately creates shares $\text{share}'_i(c_k)$ for $i = 1, \ldots, U$ using a secret sharing scheme, and (publicly) sends
$$B_{ijk} \in E'_i(\text{share}'_i(c_k)) \tag{3.15}$$

to the global authority t'_i. In order for t_j to prove that the encrypted shares A_{ij} are distributed properly (as opposed to just being random numbers), the beacon now generates K random bits. For each $k = 1, 2, \ldots, K$, one of the following is performed:

- If the kth beacon bit is 0, t_j reveals c_k and certifies that

$$C_k \in E_j(c_k). \tag{3.16}$$

  It also reveals

$$\text{share}'_i(c_k) \tag{3.17}$$

  for all i, certifying that

$$B_{ijk} \in E'_i(\text{share}'_i(c_k)). \tag{3.18}$$

  Everybody can then verify that $\text{share}'_i(c_k)$ are indeed properly constructed shares for c_k.

- If the kth beacon bit is 1, t_j reveals $c_k - \text{share}_j(Y_d)$ and certifies that

$$C_k/Z_j \in E_j(c_k - \text{share}_j(Y_d). \tag{3.19}$$

  It also reveals

$$\text{share}'_i(c_k) - \text{share}'_i(\text{share}_j(Y_d)) = \text{share}'_i(c_k - \text{share}_j(Y_d)) \tag{3.20}$$

  for all i, certifying that

$$\frac{B_{ijk}}{A_{ij}} \in E'_i(\text{share}'_i(c_k) - \text{share}'_i(\text{share}_j(Y_d)). \tag{3.21}$$

  Everybody can then verify that $\text{share}'_i(c_k - \text{share}_j(Y_d))$ are indeed properly constructed shares for $c_k - \text{share}_j(Y_d)$.

5. The global authorities now need to secretly compute $Y = \sum_{d=1}^{D} Y_d$. Before doing so, it must secretly reconstruct Y_d for each district d. Denote the "reconstruction function" for reconstructing shared secrets as R[list of intermediate authorities]. More precisely, let $R[j_1, j_2, \ldots, j_l]$ denote the unique function such that for any shared secret x among the intermediate authorities $t_{j_1}, t_{j_2}, \ldots, t_{j_l}$, the secret can be reconstructed with the identity

$$R[j_1, j_2, \ldots, j_l](\text{share}_{j_1}(x), \text{share}_{j_2}(x), \ldots, \text{share}_{j_l}(x)) = x. \tag{3.22}$$

   Here l is the exact minimum number of intermediate authorities that are required to collaborate in order to reveal the shared secret (i.e., the number k chosen in Protocol 1). The key here is that, for any given set of authorities $j_1, j_2, \ldots, j_l$, the function $R[j_1, j_2, \ldots, j_l]$ is *linear* with respect to the l-vector of shares

$$(\text{share}_{j_1}(x), \text{share}_{j_2}(x), \ldots, \text{share}_{j_l}(x)). \tag{3.23}$$

   Since each share of Y_d is available as a shared secret among the global authorities (the shares are simply A_{ij} for each global authority t'_i and each intermediate authority t_j in district d), Y_d itself can be computed as a shared secret among the global authorities. Moreover, because addition and public-scalar multiplication of shared secrets correspond homomorphically to multiplication and raising to public power of the encryptions of shared secrets, respectively, the computation of Y_d can be done by manipulating the publicly known encrypted shares A_{ij}. In this way, everybody (not just the global authorities) is able to calculate, for each global authority t'_i and each district d, the encrypted share

$$Z'_{id} \in E'_i(\text{share}'_i(Y_d)). \tag{3.24}$$

This accomplishes the goal of passing the shared secrets Y_d from the district authorities to the global authorities. If the secret sharing parameters are chosen so that more than one reconstruction of Y_d for the same district d are available (i.e., if $k < n$ in Protocol 1 at the district level), the multiple reconstructions can be checked against each other for robustness by computing the differences between pairs of reconstructions (i.e., quotients of the encrypted shares), again as shared secrets, then decrypting and comparing with zero.

6. Everybody is now able to calculate

$$Z'_i = \prod_d Z'_{id} \in E'_i(\text{share}'_i(Y)). \tag{3.25}$$

7. Each global authority t'_i now decrypts Z'_i into $\text{share}'_i(Y)$ and executes Protocol 4 to prove the decryption. Everybody can then reconstruct Y from the global shares. □

While Protocol 9 describes a hierarchical election with only one level of indirection, i.e., only one intermediate layer, it easily generalizes to multi-level hierarchical elections, i.e., elections in which there are more than one level of intermediate authorities. More specifically, such an election would start at the level of actual voters, and steps 2 through 7 of Protocol 9 would be used to verifiably and securely propagate each sub-tally up a level to higher authorities. The difference is that, in all but the highest level of the hierarchy, step 7 is not executed. This keeps all sub-tallies in intermediate layers secret. Only at the final, global level are the shares decrypted and used to reconstruct the global tally.

3.3 Real Networks

All of the previously described protocols, including ours, rely on a network topology which is complete or very close to complete. Without exception, the protocols require that participants be able to use public communication channels to broadcast data to allow all participants to verify the integrity of the election. In other words, this means that any participant can communicate with any other. While this assumption is reasonable for small-scale elections, it is less so for wide area networks, where we might have the situation that participant A's only link to all other participants goes through participant B. If B is corrupt, B can essentially gain A's vote by suppressing all of A's genuine messages and forging new ones. Clearly the problems of forging and suppression need to be addressed before our protocol can be successfully deployed on a wide area network. If these two concerns are met, then no network failure short of a partition can halt the election.

The best way of handling improper suppression of messages is to use a physical network topology which is connected enough to make it unlikely that all of one participant's links to others are corrupted. Short of this, however, we can take advantage of the fixed complexity of our protocol. While our protocol has numerous parameters that affect the number of messages which need to be sent, once these are fixed at the beginning of an election, the expected number of messages from any participant is deterministic. This prevents intermediate nodes from improperly discarding messages from other participants. Unfortunately, detection of the corrupt node must be done outside of the protocol by a real-world authority.

The obvious approach to solving the forging problem is to use authentication. If we assume the use of a public key cryptosystem such as RSA and that each participant's public key is reliably known, the forging of messages can be prevented. Some additional care must still be taken, however, to prevent replay attacks. If it can be guaranteed that each message have a unique identifier (perhaps based on election id and sequence number) then forgery attempts can be defeated as well.

What these two requirements give us is a virtual network on top of our physical one. With authenticated channels on which messages are not lost, the effective result is that of a completely connected network. Thus, robustness need not be too heavy a concern in the design of protocols over real networks.

If network communication costs are a concern, however, it is imperative that the physical topology of the real network be considered. The virtual network is complete, but the costs between nodes varies widely depending on the physical network. In dividing voters into districts in our protocol, it is (obviously) advisable to place geographically close participants in the same district.

3.4 Comparisons

As expected, our protocol (Protocol 9) makes some traditional performance for privacy tradeoffs. We analyze the differences between ours and existing protocols, primarily that of Protocol 7, below. In all of our comparisons we assume that the fraction of authorities needed to reconstruct a Shamir shared secret is p; that is, if a secret is shared among T authorities, pT of them are required to reconstruct the secret and $(1-p)T$ of them are needed to prevent reconstruction.

3.4.1 Privacy

In Protocol 7, a voter's vote is very secure. Any conspiracy of other voters gains no additional information beyond that already known by the corrupt voters unless they are able to break the encryption scheme. Only a conspiracy of pDT corrupt authorities, can compromise a voter's vote. Once this many authorities are corrupt, however, all votes are compromised.

In our protocol, a voter enjoys the same protection from other voters, but the situation with authorities becomes less clear. As in the case with Protocol 7, a voter's vote can be compromised only by corrupting enough authorities. Here, however, there are fewer authorities to corrupt; to discover a voter's vote, an adversary need only collect information from a sufficient number of authorities in a voter's district, namely pT.

More generally, our protocol is more vulnerable to an adversary which can dynamically corrupt a certain number of authorities; such an adversary can determine any individual vote it wishes. At a broader level, however, to discover every voter's vote still requires an adversary to corrupt pT authorities in each district, or pDT overall.

Naturally, if a protocol using sum secret sharing (such as Protocol 6 or subset authority voting) is used, a similar situation results. In Protocol 6 all of the authorities need to be corrupted; in subset authority voting all of the authorities in a district need to be corrupted.

3.4.2 Robustness

Here the two protocols each have their advantages. The Protocol 7 algorithm can carry on even if $(1-p)DT$ authorities simply refuse to cooperate. Once this many are corrupted, however, the entire election is invalidated and must be restarted, since not a single vote can be tallied.

In our protocol, a group of $(1-p)DT$ authorities can still prevent any votes from being tallied, but only if these are distributed evenly across districts, in such a way that no district has enough cooperative authorities. A smaller set of only $(1-p)T$ authorities in a single district is enough to prevent tallying of that district, but the cost of restarting that district's election is comparatively small. Thus our protocol is both more and less vulnerable than that of Protocol 7. On the one hand, it takes fewer corrupt authorities to damage the election, but the amount of damage is potentially smaller.

This can be seen more starkly in the comparison of Protocol 6 and subset authority voting. In Protocol 6, a single corrupt authority prevents any votes from being tallied, but in subset authority voting, a single corrupt authority disqualifies only the votes that it sees.

3.4.3 Uncoercibility

The same type of comparison holds here as with privacy and robustness. In Protocol 7, a voter's vote is uncoercible against up to pDT corrupt authorities; once again, this many corrupt authorities allows coercion of any voter. In our protocol, a voter's vote is uncoercible up to pT corrupt authorities in its district, but the effect of this coercion is limited to the district.

3.4.4 Performance

Here of course is where our protocol succeeds. Our protocol achieves savings in both computational load and message complexity.

In terms of computational load, each intermediate level authority now handles only $1/D$ the number of voters as in Protocol 7, leading to a direct factor of D reduction in computational cost. Some of this is given back by the encryption needed in our modified Protocol 8, and by the computational cost incurred by higher level authorities. Still, these should be small compared with D. The computational cost of voters also decreases, since in Protocol 7 voters are given the tasks of verifying the interactive proofs of the other voters and the final computations of the authorities, both of which scale by the number of voters. The voters then also reduce their computational load by D.

Our protocol also sends only $1/D$ as many messages as Protocol 7 does. In Protocol 7, each voter needs to broadcast to all other voters, which results in message complexity quadratic in the number of voters. In an election district with $1/D$ as many voters, only $1/D^2$ as many messages are sent. Over D election districts, this results in a total savings of $1/D$.

Chapter 4

Conclusion

In this paper, we surveyed several existing cryptographic voting protocols, and introduced a new protocol that increases scalability and reduces computation and message complexity in secure distributed elections by imposing a hierarchical structure. Our protocol handles arbitrarily complex voting trees in which votes are securely tallied at each intermediate level, localizing communication and reducing the need for global state. Just as existing voting protocols can be generalized to securely and verifiably compute arbitrary arithmetic functions, our protocol can also be used to securely and verifiably compute arbitrary expressions in distributed systems, each intermediate level handling the computation of a sub-expression.

Bibliography

[1] J. Cohen and M. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science,* pages 372–382, Portland, OR, October 1985.

[2] J. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing,* pages 52–62, Calgary, AB, August 1986.

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

[4] F. Wu. Secure Distributed Computation and Uncoercible Election Schemes. Computer science senior thesis, Harvard University, Cambridge, MA, April 1996.