Strong Privacy Protection in Electronic Voting

Jens Groth
Gorm Salomonsen

BRICS Report Series RS-04-13

ISSN 0909-0878 July 2004

Strong Privacy Protection in Electronic Voting

Jens Groth[*] Gorm Salomonsen[†]

July 19, 2004

Abstract

We give suggestions for protection against adversaries with access to the voter's equipment in voting schemes based on homomorphic encryption. Assuming an adversary has complete knowledge of the contents and computations taking place on the client machine we protect the voter's privacy in a way so that the adversary has no knowledge about the voter's choice. Furthermore, an active adversary trying to change a voter's ballot may do so, but will end up voting for a random candidate.

To accomplish the goal we assume that the voter has access to a secondary communication channel through which he can receive information inaccessible to the adversary. An example of such a secondary communication channel is ordinary mail. Additionally, we assume the existence of a trusted party that will assist in the protocol. To some extent, the actions of this trusted party are verifiable.

1 Introduction

Central to many protocols for electronic voting is the assurance of privacy. Privacy means that nobody but the voter himself knows which

[*]Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

[†]Cryptomathic

vote he cast. Voting schemes typically ensure privacy under the assumption that the client machine is uncompromised. In [DJ02], Damgård and Jurik go beyond this assumption and propose a scheme for protecting the privacy of the voter against an adversary who has full access to the client machine. Additionally, an active adversary with access to the client machine cannot cast a vote for the candidate of his choice; instead the scheme forces the adversary to vote for a random candidate. This increases the robustness of the scheme; an adversary cannot change the result of the election to something of his own wish. Unfortunately, their scheme is only practical in elections with few candidates. The goal of this paper is to propose a way of increasing privacy and robustness in elections with many candidates.

Two other notions strengthening the privacy requirement of voting schemes have been proposed in the literature: Incoercibility says that an adversary should not be able to force a voter to reveal his vote. Receipt-freeness says that the voter himself should not be able to prove to anybody how he voted. Both concepts can be demanded to various extents. In the weakest formulations, we can achieve incoercibility by erasing the memory after having cast the vote; while in the strongest formulations some physical assumptions seem to be needed [HS00]. Protection against adversaries with access to the client machine is of a somewhat different nature than incoercibility and receipt-freeness since the voter may not know that his equipment has been compromised. A scheme may for instance achieve incoercibility and/or receipt-freeness by enabling the voter to produce a false, but convincing, transcript of his computations that he can show to the coercer. In contrast to this, we assume that the adversary has full access to the client machine on which an unsuspecting voter is producing a vote; therefore, the adversary does have access to the correct transcript of the computations going on.

In an Internet voting protocol, the voter uses a client machine to enter his vote and transmit it over the Internet. There are many realistic adversaries that could have access to the client machine, for instance hackers and system administrators. In addition, due to disk swapping, etc., it is not always the case that data is erased. A subsequent user of the client machine may therefore be able to see what has occurred in prior sessions.

The idea in [DJ02] for protection against adversaries with access to the client machine is the following: Each candidate is represented by a number $0 \leq j < L$, where L is the number of candidates. A trusted party selects for each voter a permutation π. By secondary means, for instance through ordinary mail, the voter receives a ballot with the candidates and their permuted numbers, $\pi(0), \ldots, \pi(L-1)$. He enters $\pi(j)$ to vote for candidate j. The client machine encodes and encrypts $\pi(j)$ and sends it to the election authorities. This can be seen as a combination of two means to protect the voter's privacy. An adversary without access to the secondary channel does not know which candidate is represented by $\pi(j)$. On the other hand, an adversary with access to the secondary channel but without access to the client machine only sees an encrypted message and has no clue about the vote. Realistic adversaries will not have access to both the client machine and the secondary channel at the same time, and therefore this combination protects voter privacy in a stronger sense than what is accomplished by voting protocols.

In [DJ02], the basic voting scheme is based on a homomorphic public-key threshold cryptosystem. The problem when receiving a vote $E_{pk}(\pi(j))$ where the chosen candidate is permuted is to create a correct encrypted vote $E_{pk}(j)$ on the candidate, in other words to invert the permutation under the encryption. They propose a multi-party computational method for doing this, but unfortunately it is, even if the trusted party helps, not efficient enough to handle elections with many candidates.

We have a simple idea to speed up computations. Instead of giving each voter a random permutation, we assign the same permutations to groups of voters. Instead of converting each individual vote, we then try to convert entire groups of votes. The vote conversion protocol in [DJ02] does not work with groups of voters so we have to invent a new protocol for doing so. Additionally, we must now manage the groups of voters so that the adversary still cannot link the individual voters to their permutations.

2 Background Information

2.1 Voting Scheme

We look at elections where M is a strict upper bound on the number of voters and L is the number of candidates or options, possibly including dummy candidates for blank votes, etc. A vote on candidate j, where $0 \leq j < L$, is represented by the number M^j. To vote on such a candidate the voter in the basic scheme encrypts M^j and sends it to the authorities together with a zero-knowledge proof of knowledge that the content of the ciphertext is a legitimate vote on some candidate.

We assume the parties have access to an authenticated broadcast channel with memory. We imagine this as a message board where each party has a segment where only he can write, for instance implemented through digital signatures, and nobody can erase anything. This means that each participant in the protocol can post a message in a manner so everybody is assured of the sender's identity and that everybody else has received the same message. The voters send their encrypted votes to the authorities by posting them on the message board. This way everybody can check that only eligible voters cast a vote, and that those voters cast at most one vote.

For encryption, a homomorphic public-key threshold cryptosystem is used. By homomorphic we mean $E_{pk}(M^i + M^j) = E_{pk}(M^i)E_{pk}(M^j)$. This means that by multiplying the valid encrypted votes, $E_1, \ldots, E_{M'}$ with the number of votes $M' < M$, we get an encryption of $\sum_{j=0}^{L-1} v_j M^j$, where v_j is the number of votes on candidate number j. Several voting schemes of this type have been suggested and they look very promising for real life application, see for instance [CGS97],[DJ01],[BFP+01] and [DGS03].

To protect privacy we secret share the private key between N authorities using a (t, N) threshold scheme. This means that less than t authorities cannot decrypt ciphertexts, while t cooperating authorities are capable of decrypting ciphertexts. Each individual voter's vote therefore remains secret unless t authorities unite. To compute the result of the election the authorities cooperate to decrypt the product of all the ciphertexts corresponding to valid votes. In the end, they broadcast the result of the election.

2.2 Extras Needed for Strong Privacy

In order to protect the voter using a monitored piece of equipment we must give him some extra information not present on the machine. We therefore assume some kind of secondary channel from which he can get some input from the authorities managing the election. This can for instance be a paper ballot sent to the voter by ordinary mail, be some information given to the voter when registering for the election, or be information he receives over his mobile phone. We assume that the adversary does not have access to both this channel and the client machine at the same time.

When assuming a secondary channel to the voter it seems like in most reasonable scenarios there is a single entity having knowledge of the voter's information. Certainly, we can think of schemes that remedy this deficiency, for instance, where the voter receives several ballots that have been printed in different locations, however, such measures go beyond the scope of this paper. We therefore assume there is a single party that is trusted to create and deliver some secret information to each voter. In terms of privacy, this single party is not trusted; the protocol will provide privacy as long as the adversary does not control both the client machine and the single party. On the other hand, a malicious single party may be able to tamper with the correctness of the result. So what we are presenting is actually a privacy/correctness tradeoff.

2.3 Homomorphic Cryptosystems and Integer Commitment Schemes

As mentioned above we use a public-key homomorphic threshold cryptosystem for the election. There is a public key pk published on the message board for all to see. Furthermore, there are N authorities sharing the private key. The message space of the election is $\mathbf{Z}_n$ for some integer $n \geq M^L$. Ciphertexts belong to a group that we write multiplicatively. The homomorphic property of the cryptosystem is the following: If we have two plaintexts m_1, m_2 encrypted with randomness r_1, r_2 as $E_1 = E_{pk}(m_1; r_1)$ and $E_2 = E_{pk}(m_2; r_2)$ then $E_1 E_2 = E_{pk}(m_1 + m_2; r_1 + r_2)$. An example of such a cryptosystem is the Paillier style cryptosystem from [DJ01].

We also use a homomorphic integer commitment scheme. Also for this scheme, a public key K is published. The message space is the entire set of integers and the commitment scheme is homomorphic in the sense that for $c_1 = com_K(m_1; r_1)$ and $c_2 = com_K(m_2; r_2)$ we have $c_1 c_2 = com_K(m_1 + m_2; r_1 + r_2)$. An example of such a homomorphic integer commitment scheme is given in [DF02].

3 Managing the Groups of Voters

The idea in our scheme is to have a trusted third party that divides the voters into disjoint groups $S_1, \ldots, S_Q$. When votes are tabulated, we pool together the votes from one group and handle the entire bundle in the same manner. Having divided the voters into groups we face the problem of combining the votes in the groups in a way that does not reveal to which group each individual voter belongs. Revealing the voter's group affiliation would weaken the security of the scheme since an adversary might then learn the permutation associated with said voter.

For this purpose, a verifiable secret shuffle seems like the right choice. Such a scheme allows the trusted party to re-encrypt all the votes, permute them, and prove in zero-knowledge that indeed he has made such a permutation. Efficient verifiable secret shuffles have been suggested in [FS01, Nef01, Gro03]. The latter allows us to use most of the known homomorphic cryptosystems. They have the additional advantage that in a natural way the trusted party can commit to the permutation before receiving any encrypted votes. We do the division of the voters by letting the first group be the first $|S_1|$ ciphertexts coming out of the shuffle, the second group is the following $|S_2|$ ciphertexts from the shuffle, etc. We can use the convention that voters not having submitted a vote automatically are assigned the vote $E_{pk}(0;0)$, which will not affect the outcome of the election but will ensure that their group membership remains secret.

Let us look quickly at the shuffle scheme in [Gro03] to be more precise and to describe the few modifications of it we need for our purpose. Given ciphertexts $E_1, \ldots, E_{M'}$ the goal is to shuffle them according to a permutation ψ into a new set of ciphertexts $E'_1, \ldots, E'_{M'}$ so that the corresponding plaintexts $m_1, \ldots, m_{M'}$ and $m'_1, \ldots, m'_{M'}$ satisfy $m'_1 = m_{\psi(1)}, \ldots, m'_{M'} = m_{\psi(M')}$. We do this in two steps. First the trusted party commits to $\psi(1), \ldots, \psi(M')$, in that order. By making the commitments public, he essentially commits to the permutation ψ of the voters. The first step can be done independently of the actual ciphertexts without compromising the security of the shuffle scheme, and if need be it is possible already at this stage to prove in special honest verifier zero-knowledge that he has committed to a permutation of the voters. In the next step the trusted party receives the ciphertexts $E_1, \ldots, E_{M'}$ and re-encrypts and permutes them by setting $E'_1 = E_{\psi(1)}E_{pk}(0), \ldots, E'_{M'} = E_{\psi(M')}E_{pk}(0)$. Finally, he proves that the commitments to $\psi(1), \ldots, \psi(M')$ were correctly formed and that he has shuffled the encryptions according to the same secret permutation ψ.

4 Inverting Permutations

So far we have the following components of a protocol: The trusted party can organize the voters into groups $S_1, \ldots, S_Q$, select and distribute to the voters in these groups permutations $\pi_1, \dots, \pi_Q$, and when receiving the encrypted votes shuffle those ciphertexts into place such that the first $|S_1|$ shuffled ciphertexts are those corresponding to voters with permutation π_1, etc.

We want a method to transform the ciphertexts where votes are permuted under some permutation π_i into something that can be used in the basic voting protocol, i.e., an encryption of votes that are not permuted. Since we are just focusing on one such group let us simplify notation by calling the relevant permutation π, say that there are T voters in the group with corresponding ciphertexts $E_1, \dots, E_T$. We proceed by computing the product of the ciphertexts, giving us a ciphertext E encrypting $\sum_{j=0}^{L-1} v_j M^{\pi(j)}$, where v_j is the number of votes on candidate j. We shall provide a multi-party computation protocol for the authorities to transform E into a new ciphertext E' encrypting $\sum_{j=0}^{L-1} v_j M^j$. We first assume that π is known. Later in the section, we shall investigate the case where π is unknown.

We invert the permutation in two steps. First the authorities create an encryption E_{R_π} of a number $R_\pi = \sum_{j=0}^{L-1} R_j M^{\pi(j)}$. The numbers R_j must for each j be chosen so that $R_j + v_j < M$. We will reveal $R_j + v_j$ and the purpose of R_j is to hide v_j. At the same time they produce E_R as an encryption of $R = \sum_{j=0}^{L-1} R_j M^j$. In the second step the authorities decrypt $E_{R_\pi} E$ to get the plaintext $\sum_{i=0}^{L-1}(R_j + v_j) M^{\pi(j)}$. They let E' be $E_R^{-1} E_{pk}(\sum_{i=0}^{L-1}(R_j + v_j) M^j; 0)$, containing the wanted plaintext $\sum_{j=0}^{L-1} v_j M^j$.

The crucial point is to generate E_{R_π} and E_R in a distributed way. A possibility is the following: For each $i = 1, \dots, N$, authority i selects $R_{i,0}, \dots, R_{i,L-1}$ at random from $\{0, \dots, \lfloor \frac{M-T}{N} \rfloor\}$ and generates an encryption $E_{R_\pi,i} = E_{pk}(\sum_{j=0}^{L-1} R_{i,j} M^{\pi(j)})$. This way $E_{R_\pi} = \prod_{i=1}^{N} E_{R_\pi,i}$ will have the required properties. Similarly each authority generates $E_{R,i} = E_{pk}(\sum_{j=0}^{L-1} R_{i,j} M^j)$, and E_R can be computed as $\prod_{i=1}^{N} E_{R,i}$. Each authority can prove in zero-knowledge that it has generated $E_{R_\pi,i}$ and $E_{R,i}$ correctly by making integer commitments $c_{i,0}, \dots, c_{i,L-1}$ to $R_{i,0}, \dots, R_{i,L-1}$, use range proofs as in [Bou02] to show that they are in the correct interval, and use equivalence proofs to show that $E_{R_\pi,i}$ has the same content as $\prod_{j=0}^{L-1} c_j^{M^{\pi(j)}}$ and $E_{R,i}$ has the same content as $\prod_{j=0}^{L-1} c_j^{M^j}$.

As an alternative to showing the permutation in open, the trusted party may also select for each group of voters a hidden permutation. A permutation for a group can be provided by the trusted party through making ciphertexts $E_{\pi,0} = E_{pk}(M^{\pi(0)}), \ldots, E_{\pi,L-1} = E_{pk}(M^{\pi(L-1)})$ and $E_{\pi^{-1},0} = E_{pk}(M^{\pi^{-1}(0)}), \ldots, E_{\pi^{-1},L-1} = E_{pk}(M^{\pi^{-1}(L-1)})$ public. The tally servers may produce E_R in the same way as they did above. When producing E_{R_π} they form $E_{R_\pi,i}$ in a different way. Tally server i still uses the commitments $c_{i,0}, \ldots, c_{i,L-1}$ in the proof of correctness of E_R in the same way as above, however, this time it forms $E_{R_\pi,i}$ as $E_{pk}(0) \prod_{j=0}^{L-1} E_{\pi,j}^{R_{i,j}}$, and uses multiplication proofs to demonstrate that this ciphertext has been correctly formed.

After receiving the votes the tally servers have to create an encryption of $\sum_{i=0}^{L-1}(R_j + v_j)M^j$. Since $\sum_{j=0}^{L-1}(R_j + v_j)M^{\pi(j)} = \sum_{j=0}^{L-1}(R_{\pi^{-1}(j)} + v_{\pi^{-1}(j)})M^j$ is revealed we can form $E_R^{-1} \prod_{j=0}^{L-1} E_{\pi^{-1},j}^{R_{\pi^{-1}(j)}+v_{\pi^{-1}(j)}}$ to get the required ciphertext E' encrypting $\sum_{j=0}^{L-1} v_j M^j$.

Of course when making the ciphertexts this way the tally servers need assurance that $E_{\pi,0}, \ldots, E_{\pi,L-1}$ and $E_{\pi^{-1},0}, \ldots, E_{\pi^{-1},L-1}$ correspond to a hidden permutation. This can be proved in zero-knowledge by running a shuffle proof twice. We now show that $E_{\pi^{-1},0}, \ldots, E_{\pi^{-1},L-1}$ shuffles into $E_{\iota,0} = E_{pk}(M^0; 0), \ldots, E_{\iota,L-1} = E_{pk}(M^{L-1}; 0)$, and that $E_{\iota,0}, \ldots, E_{\iota,L-1}$ shuffles into $E_{\pi,0}, \ldots, E_{\pi,L-1}$ using the same permutation as in the first shuffle.

5 Analysis of the Protocol

5.1 Privacy

The main purpose of the protocol is to strengthen privacy. Suppose we divide the voters into L groups and assign them the permutations $\pi_1, \ldots, \pi_L$, where $\pi_i(j) = j + i \bmod L$. The adversary does not know to which group a voter belongs, unless a huge amount of voters has been corrupted. Therefore, on seeing $\pi(j)$ he has no knowledge about j.

The protocol is an add-on to the standard voting protocols based on homomorphic encryption. This means, even if the trusted party that creates permutations and distributes them is dishonest, the privacy protection of the standard protocol is intact and protects the voter's privacy. Only when the adversary has access to both the secondary channel and the client machine can he compromise the privacy of the voter.

5.2 Correctness

The proposed method can also hamper, somewhat, attackers that try to modify the result of an election. Regarding the latter we achieve, when the protocol works at its best, that an attacker can submit only a random vote on some other candidate than the one chosen by the voter. Obviously, this is not ideal since in the real world votes are usually not distributed equally between candidates. However, it is better than nothing.

We note that a little trick can be deployed to see whether an election has been conducted without a massive attack on the robustness of the election. The trick consists in creating some dummy candidates that cannot be chosen by honest voters. If the result shows votes on these candidates then some sort of cheating has occurred. We cannot differentiate the types of cheating though. It may be because hackers have attacked and thus some votes had been cast at random. It may also be a group of discontent voters that try to make it look like an attack by hackers has taken place.

Let us look at the case where the voters only have a moderate number of candidates to choose from and may only cast one vote. In this case we may select a family of permutations P, so that for any two pairs of candidates $(i, j), (a, b)$ where $i \neq j$ and $a \neq b$, the probability when choosing π at random from P for $\pi(i) = a, \pi(j) = b$ is $\frac{1}{L(L-1)}$. If a voter holds a random permutation from P and the adversary does not know this permutation, then the adversary has no idea of the voters choice i even when seeing $a = \pi(i)$. Furthermore, if he chooses $b \neq \pi(i)$ then he simply votes for a random candidate $j \neq i$. As an example of such a family of permutations we may if L is a power of a prime interpret the candidates as elements in a finite field of order L and let the family of permutations be the $L(L-1)$ non-constant lines in the field.

The scheme above can be used with known permutations dividing voters into $|P|$ groups of equal size and assigning each of the groups a permutation from P. The adversary still does not know to which group a voter belongs. However, when the number of candidates is large this is not a practical approach. We may decide to reduce the number of permutations in the family. To protect privacy we only need L permutations, but an attacker having some idea of a voter's preference may then cheat with that vote. As an alternative, we can hide the permutations using the protocol with hidden permutations. Certainly, a determined attacker may collect ballots from voters to get a picture of the permutations in play; however, this will require a huge effort. Likewise, an adversary might corrupt some of the authorities and through their choices of R's used to hide the election outcomes in the groups obtain some statistical information about the permutations, but again this requires much effort from the adversary with little success to be expected.

We can imagine voting schemes where the voters may cast multiple votes at once. In other words they submit a vote on the form $\sum_{j=0}^{L-1} \delta_j M^j$ where $\delta_j = 1$ for the candidates selected and $\delta_j = 0$ otherwise. Again, this may necessitate hiding the permutations. Otherwise, an attacker might be able to detect certain patterns in the voter's choice and correlate that with the permutations in play to determine the choice. Having an idea for instance that a particular voter probably intends to vote for candidate 1,5,6 and 9 and seeing numbers 2,3,4,10 he might find that there is indeed a permutation π so that $\pi(1) = 2, \pi(6) = 3, \pi(5) = 4$ and $\pi(9) = 10$. This would give him good reason to believe that he had guessed the voter's choice correctly.

5.3 Power of the Trusted Party

Since we rely on a trusted party to perform some of the operations in the protocol, it is relevant to consider how much trust we have to place in this party. First, we note that what we do here is to give an extra guarantee of privacy. No matter how the trusted party may try to cheat, the voter's privacy protection under the original basic voting scheme is still effective. But of course, with a cheating third party the extra guarantees we try to provide against adversaries with access to the client machine no longer hold.

The voting protocol itself is still used for verification of the validity of the votes. Furthermore, the trusted party does have to prove the correctness of the shuffle. Therefore, the trusted party cannot add votes or remove votes. The only possible cheating left consists in sending the voter an invalid ballot. If the voter receives an invalid ballot, he may this way be tricked into voting for another candidate than he wishes to vote for. Moreover, since the voter in this protocol has a personal ballot there is no public information available enabling him to discover the problem. However, we may imagine that the voter can request from the trusted party an opening of the commitment indicating in which group he is to be placed. In the open permutation protocol, he can this way directly see whether his ballot matches the permutation of his group. Of course, this method only works in scenarios where the adversary with control over the client machine is kept from sending in such a request. We can remedy this latter deficiency with another method, namely giving the voter two ballots and having him indicate, publicly, which ballot he is using. We then require by default that the trusted party open the ballot the voter has not used. This cut-and-choose protocol limits the possibilities for the trusted party to cheat. A cheating trusted party is then going to be caught with high probability when sending out more than a fraction of false ballots.

5.4 Efficiency

Our scheme does not alter the efficiency of the voting scheme on the client side. For the voter the extra privacy protection comes for free. On the server side, we compare the efficiency of our scheme with that of [DJ02]. Both in their article and in ours we have formulated the schemes in broad terms of some homomorphic cryptosystem, etc. However, no matter which operation is the most expensive one in their scheme, they require the servers to perform $\Theta(ML)$ operations each and the trusted party to perform $\Theta(ML)$ operations. In comparison, in our scheme in both the known permutations scenario and the hidden permutations scenario the servers use $\mathrm{O}(QL+M)$ operations each, and the trusted party uses $\mathrm{O}(QL+M)$ operations. Looking at the schemes using, say, the generalized Paillier encryption of [DJ01] for encryption and the integer commitment scheme from [DF02] it also seems like the constants in their protocol are higher than ours. Taking as a toy example an election with 1,000,000 voters and 101 candidates, we can save at least a factor 100 compared to their scheme. This is enough to make our scheme practical.

References

[BFP$^+$01] Oliver Baudron, Pierre-Alain Fouque, David Pointcheval, Guillaume Poupard, and Jacques Stern. Practical multi-candidate election scheme. In *proceedings of PODC '01*, pages 274–283, 2001.

[Bou02] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *proceedings of EUROCRYPT '00, LNCS series, volume 1807*, pages 431–444, 2002.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally eficient multi-authority election scheme. In *proceedings of EUROCRYPT '97, LNCS series, volume 1233*, pages 103–118, 1997.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *proceedings of ASIACRYPT '02, LNCS series, volume 2501*, pages 125–142, 2002.

[DGS03] Ivan Damgård, Jens Groth, and Gorm Salomonsen. The theory and implementation of an electronic voting system. In D. Gritzalis, editor, *Secure Electronic Voting*, pages 77–100. Kluwer Academic Publishers, 2003.

[DJ01] Ivan Damgård and Mads J. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *proceedings of PKC '01, LNCS series, volume 1992*, 2001.

[DJ02] Ivan Damgård and Mads J. Jurik. Client/server tradeoffs for online elections. In *proceedings of PKC '02, LNCS series, volume 2274*, 2002.

[FS01] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *proceedings of CRYPTO '01, LNCS series, volume 2139*, pages 368–387, 2001.

[Gro03] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In *proceedings of PKC '03, LNCS series, volume 2567*, pages 145–160, 2003.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *proceedings of EUROCRYPT '00, LNCS series, volume 1807*, pages 539–556, 2000.

[Nef01] Andrew C. Neff. A verifiable secret shuffle and its application to e-voting. In *CCS '01*, pages 116–125, 2001. Full paper available at http://www.votehere.net/vhti/documentation/egshuf.pdf.