# Receipt-Free Mix-Type Voting Scheme
## –A practical solution to the implementation of a voting booth–

Kazue Sako[1] and Joe Kilian[2]

[1] NEC Corporation, 4-1-1 Miyazaki Miyamae, Kawasaki 216, JAPAN
[2] NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA

**Abstract.** We present a receipt-free voting scheme based on a mix-type anonymous channel[Cha81, PIK93]. The receipt-freeness property [BT94] enables voters to hide how they have voted even from a powerful adversary who is trying to coerce him. The work of [BT94] gave the first solution using a *voting booth*, which is a hardware assumption not unlike that in current physical elections. In our proposed scheme, we reduce the physical assumptions required to obtain receipt-freeness. Our sole physical assumption is the existence of a private channel through which the center can send the voter a message without fear of eavesdropping.

## 1 Introduction

### 1.1 Receipt-Free Voting Schemes

The ultimate goal of secure electronic voting is to replace physical voting booths. Achieving this goal requires work both on improving the efficiency of current protocols and understanding the security properties that these physical devices can provide. Recently, Benaloh and Tuinstra[BT94] observed that, unlike physical voting protocols, nearly all electronic voting protocols give the voters a receipt by which they can prove how they voted. Such receipts provide a ready means by which voters can sell their votes or another party can coerce a voter. Benaloh and Tuinstra give the first *receipt-free* protocol for electronic voting. In their scheme a trusted center generates for each voter a pair of ballots consisting of a "yes" vote and a "no" vote in random order. Using a trusted beacon and a physical voting booth the center proves to the public that the ballot indeed consists of a well-formed (yes/no) or (no/yes) pair and at the same time proves to the verifier which pair it is. The physical apparatus ensures that by the time the verifier is able to communicate with an outsider, he can forge a proof that the ballot is (yes/no) and also forge a proof that it is (no/yes). Thus, such a proof ceases to provide either proof as a receipt.

Independently, Niemi and Renvall[NR94] tried to solve this problem. They also use a physical voting booth where a voter performs multiparty computation with all the centers.

Both the Benaloh-Tuinstra and the Niemi-Renvall protocols illustrate that receipt-freeness is possible. However, their physical requirements are fairly cumbersome, and are not unlike those faced by participants in physical elections. An important open question is precisely what physical requirements are necessary for achieving receipt-freeness.

### 1.2 Results of This Paper

In this paper we consider how to implement receipt-freeness in a more practical manner. We start with the mix-type protocols of [Cha81, PIK93] and augment them to obtain a protocol which is receipt-free and *universally verifiable*. By universally verifiable we mean that in the course of the protocol the participants broadcast information that allows any voter or interested third party to at a later time verify that the election was properly performed. To make our protocol receipt-free we must by necessity make some physical assumption. We assume the existence of an untappable private channel. Our untappability requirement is physical; cryptographic implementations of untappable channels do not suffice for our purposes. To obtain universal verifiability we develop efficient techniques by which a mixer can prove that they performed correctly, and use the Fiat-Shamir [FS86] technique to make these proofs noninteractive.

### 1.3 Techniques Used

**Chameleon blobs**
Brassard, Chaum and Crépeau introduced the concept of zero-knowledge proofs and zero-knowledge bit-commitment schemes[BCC88]. In a zero-knowledge bit commitment scheme the prover commits to $b$ by generating a pair $(B, S_b)$ ($B$ is referred to as a *blob*) and sends $B$ to the verifier. Later, the prover can *open* a blob by sending $S_b$ verifier, who evaluates $\mathsf{open}(B, S_b)$ to obtain $0, 1$ or **reject**. If the prover behaves properly then $\mathsf{open}(B, S_b) = b$. The distribution on $B$ is independent of $b$, however a computationally bounded prover cannot generate a triple $(B, S_0, S_1)$ such that $\mathsf{open}(B, S_b) = b$ for $b \in \{0, 1\}$. That is, once a prover has committed to a bit with $B$, he can open it only one way. A system of *chameleon blobs* is a system with the additional property that the verifier can, on input $(B, b)$ generate $S_b$ such that $(B, S_b)$ evaluates to $b$. That is, the verifier knows how to open a blob both ways. Furthermore, we require that the conditional distribution on $S_b$ given $B$ be the same as the conditional distribution generated by $P$. We use chameleon blobs to allow the verifier to forge proofs.

**Amortization techniques**
In order to achieve universal verifiability, we require the mixers to prove that they are not altering the ballots. These proofs greatly increase the communication complexity of the protocol. To ameliorate this problem, we show how to use techniques similar to those used in [SK94] to reduce the amount of communication and computation necessary to generate, transmit and check the proofs.

### 1.4 Outline of the Rest of the Paper

In Section 2 we construct a mix-net with the universal verifiability property. In Section 3 we give a receipt-free voting scheme.

## 2 Universally Verifiable Mix-Net

Mix-net anonymous channels were first proposed by [Cha81]. Subsequently, many voting schemes have been proposed based on this basic technique [FOO92, PIK93]. However, this type of scheme has only *individual verifiability*. That is, a sender can verify whether or not his message has reached its destination, but cannot determine if this is true for the other voters. A disadvantage of this situation is that one has to trust other voters to be vigilant in checking that their vote was counted. Also, one may wish to audit an election, checking that is was fair, without getting back in touch with all of the voters. Thus, universal verifiability is preferable to individual verifiability, provided that it is not too expensive.

In this section, we describe a scheme for mix-net proposed by [PIK93], and give a protocol to make the scheme universally verifiable. Furthermore, we show how to amortize the cost for some of the verification procedures required by our scheme.

### 2.1 A Scheme with Individual Verifiability

The paper of [PIK93] gives two types of mix-type anonymous channels. Both types of schemes achieve only individual verifiability; we add additional protocols to achieve universal verifiability.

We first outline (with slightly modifications) the anonymous channel protocol proposed in [PIK93]. In this scheme, encrypted messages from the senders are successively processed by the mixing centers until the last center outputs a randomly, untraceably ordered set of unencrypted[3] At a high level, the senders first post their encrypted messages. Center $i$ processes each message posted by Center $i-1$ (or the senders, when $i=1$) and posts the results in permuted order. It remains to specify how a message $m$ is initially encrypted by a sender and how Center $i$ processes each message.

In the following, the definition of the "generating element" $g$ is modified from the original scheme, in order to evade an attack proposed by Pfitzmann [Pfi94].

A Mix-Type Anonymous Channel by [PIK93]

Public information : $p = kq + 1$ ($p, q$ prime),
$g = (g')^k \bmod p$( where $g'$ is a generator mod$p$ )
Public key of center $i$ : $y_i = g^{x_i} \bmod p$
Secret key of center $i$ : $x_i \in Z_q^*$
Message from the sender : $m$
We define $w_i = y_{i+1}y_{i+2}\cdots y_n$ and $w_n = 1$.

[3] That is, the encryptions used for the anonymous channel have been stripped off. Of course, these messages may have been encrypted before being sent through the channel.

**Encrypting a message**
The sender generates a random number $r_0$, and posts

$$Z_1 = (G_1, M_1) = (g^{r_0} \bmod p, \quad (w_0)^{r_0} \cdot m \bmod p)$$

for use by Center 1.

**Processing a message**
On input $(G_i, M_i)$, Center $i (i = 1, \cdots, n-1)$ generates a random number $r_i$ (independently for each message-pair) and calculates the following using his secret key $x_i$:

$$\begin{aligned} G_{i+1} &= G_i \cdot g^{r_i} \mod p \\ &= g^{r_0 + \cdots + r_i} \mod p \\ M_{i+1} &= M_i \cdot w_i^{r_i} / G_i^{x_i} \mod p \\ &= w_i^{r_0 + \cdots + r_i} \cdot m \mod p \end{aligned}$$

He posts $Z_{i+1} = (G_{i+1}, M_{i+1})$ (permuted with the other processed messages) for use by Center $i+1$.

Center $n$ recovers $m$ by computing

$$m = M_n / G_n^{x_n} \bmod p.$$

By adding redundancy to the message $m$, and by having the last center $n$ announce all the received messages (again in permuted order), a sender can check whether or not his message has reached the destination. However, this gives only individual verifiability, as a sender can not directly determine if the other messages have been properly handled. Also, the redundancy can be used as a receipt, precluding the possibility of receipt-freeness.

## 2.2 Achieving Universal Verifiability

We obtain universal verifiability in the above scheme, and the other scheme discussed in their paper, by requiring each center to prove that they correctly processed their messages. At this time, or later, any interested party can check the resulting proofs to confirm that the messages have all been handled correctly. With this method for achieving universal verifiability there is no need for adding redundancy to the messages. Furthermore, it also helps thwart an attack proposed in [Pfi94].

We first modify the way each center processes the pairs. Given a pair $(G_i, M_i)$, a center computes $(G_{i+1}, M_{i+1})$, but in the first phase it posts $G_i^{x_i}$. In the second phase it then posts the pairs $(G_{i+1}, M_{i+1})$ in permuted order. Note that this protocol leaks the value of $G_i^{x_i}$, which was not leaked in the original protocol. We know of no way to exploit this extra information.

A center proves the correctness of each stage separately. We write these protocols in terms of an interactive proof system; they may then be made non-interactive using the Fiat-Shamir technique [FS86].

**Proving correctness for the first phase**

We can abstract the first phase of the protocol as follows. Given $G$, the first phase consists of performing decryption and generate $H = G^x \bmod p$. The proof consists of, given $(G, g, y = g^x \bmod p)$, showing that $H$ is generated in this manner from $G$.

prove DECRYPT

**1.** The prover uniformly chooses $r \in Z_{p-1}$. Let

$$y' = g^r \bmod p$$
$$G' = G^r \bmod p.$$

The prover sends $(y', G')$.

**2a.** With probability $\frac{1}{2}$, the verifier asks the prover to reveal $r$. The verifier checks that $y'$ and $G'$ are consistent with $r$.

**2b.** With probability $\frac{1}{2}$, the verifier asks the prover to reveal $r' = r - x$. The verifier checks that

$$y' = g^{r'} \cdot y \bmod p \text{ and}$$
$$G' = H \cdot G^{r'} \bmod p.$$

**Proving correctness for the second phase**

We may slightly abstract the second phase as follows.

Given constants $g, w$ and

$$A = \begin{pmatrix} a_i^{(1)} \\ a_i^{(2)} \end{pmatrix},$$

the second phase consists of generating $r_1, r_2, \ldots$ and a permutation $\pi$ and generating a set of pairs

$$B = \begin{pmatrix} a_{\pi(i)}{}^{(1)} \cdot g^{r'_{\pi(i)}} \bmod p \\ a_{\pi(i)}{}^{(2)} \cdot w^{r'_{\pi(i)}} \bmod p \end{pmatrix}.$$

Here $a_i^{(1)}$ refer to $G$'s and $a_i^{(2)}$ to $M/H$'s in the first phase. The proof consists of, given $(A, B, g, w)$, showing that $B$ could be generated in this manner from $A$.

prove SHUFFLE

**1.** The prover uniformly chooses $t_i \in Z_{p-1}$, random permutation $\lambda$ and

$$C = \begin{pmatrix} a_{\lambda(i)}{}^{(1)} \cdot g^{t_{\lambda(i)}} \bmod p \\ a_{\lambda(i)}{}^{(2)} \cdot w^{t_{\lambda(i)}} \bmod p \end{pmatrix}.$$

The prover sends C.

**2a.** With probability $\frac{1}{2}$, the verifier asks the prover to reveal $\lambda$ and $t_i$. The verifier checks that $C$ is consistent with $A, \lambda, t_i$ in that way.

**2b.** With probability $\frac{1}{2}$, the verifier asks the prover to reveal $\lambda' = \lambda \circ \pi^{-1}$ and $t'_i = t_i - r'_i$. The verifier checks that $C$ can be generated from $B$ in the following way: For

$$B = \begin{pmatrix} b_i^{(1)} \\ b_i^{(2)} \end{pmatrix},$$

$$C = \begin{pmatrix} {b_{\lambda'(i)}}^{(1)} \cdot g^{t'_{\lambda'(i)}} \bmod p \\ {b_{\lambda'(i)}}^{(2)} \cdot w^{t'_{\lambda'(i)}} \bmod p \end{pmatrix}$$

holds.

### 2.3 Processing Multiple Messages Together

In this section, we show that the centers can process multiple messages together to achieve a reduces amortized cost per message. Instead of executing "prove DECRYPT" protocol for each shuffled component, a center can prove a single statement equivalent to proving that he decrypted all the components correctly in the following way.

We need to show the following equation holds for each component $i$.

$$H^{(j)} = (G^{(j)})^x \bmod p$$

We can reduce above equations to the following one equation using randomly chosen coefficients $c_i$.

$$\prod_i (H^{(j)})^{c_i} = \prod ((G^{(j)})^{c_i})^x \bmod p$$

The center can execute above protocol where $G = \prod_i (G^{(j)})^{c_i}$ and $H = \prod (H^{(j)})^{c_i}$. We exploit the fact that if one or more of the original equations is wrong then if the coefficients are chosen randomly the final equation will also be wrong. Note that these coefficients must not be picked by the prover, but should be given by a verifier, beacon or as the output of a suitable hash function.

### 2.4 Remarks on Vote Duplication

Gennaro [Gen94] has pointed out that in the Sako-Kilian [SK94] voting protocol a malicious voter may copy another voter's vote by simply duplicating the ballot. This attack is readily foiled by a simple modification to the Fiat-Shamir heuristic; see [Gen94] for a number of fixes. However, we note that for some of the mix-type voting schemes proposed in the literature the problem is even more severe. First, in the usual mix-type voting paradigm the ballots have redundancy attached to allow voters to check that their vote has been counted. An attacker may then duplicate a ballot and search the published list of received votes to find the two identical ballots, revealing how that entity voted.

Simple vote duplication may be easily detected by noticing the duplicate ballots in the first stage, and the adversary risks identification as well as detection. A more subtle attack on the [PIK93] scheme (which may therefore also be applied to our current scheme) involves "blinding" a duplicate ballot so that it looks different but eventually yields the same ballot in the end. If the legitimate voter casts an encrypted ballot of the form

$$Z_1 = (G_1, M_1) = (g^{r_0} \bmod p, \quad (y_1 \cdots y_n)^{r_0} \cdot m \bmod p),$$

an attacker can choose $r \in Z_p^*$ at random and cast the ballot

$$Z_1' = (G_1', M_1') = (G_1 \cdot g^{r'} \bmod p, \quad M_1 \cdot (y_1 \cdots y_n)^{r'} \bmod p),$$

giving no obvious relationship between $Z_1$ and $Z_1'$. At the end of the anonymous channel protocol, the final center can still detect vote duplication and refuse to reveal these votes, but then it is more difficult for others to be assured that the last center isn't just trying to impede the election.

We note that in our scheme there is no need for extra redundancy. Hence, even a successful vote duplication attack only gives indirect statistical information to the attacker, and is not useful when the number of votes for each choice is large. One approach for evading this problem entirely is to have $Z_1$ signed and encrypted using the first center's public key. Unless the adversary colludes with the first center, he would not succeed in copying the vote. Even if he does succeed in copying, the inference of a vote may be excluded by omitting the redundancy in each messages. Guarding against a colluding center is more difficult. One approach, is to have a two-step process whereby the voters commit to their first posting and then reveal it. This is somewhat against the spirit of the principle that a voter can vote and then walk away. However, this is not such a big deal when there are a small number of voters, which is precisely the case where such attacks are most troublesome.

## 3 Proposed Receipt-Free Scheme

In this section, we describe a mix-type receipt-free voting scheme. Subsection 3.1 gives an overview of the scheme and Subsection 3.2 gives a more detailed description. We note that the interactive zero-knowledge proofs can be made non-interactive by again using Fiat-Shamir technique [FS86].

Our assumptions are as follows: First, need a physically untappable means of communication between the mixing centers and the voters. By a standard exclusive-or trick this assumption can be implemented by having a number of communication channels, assuming that the adversary can't simultaneously tap every one of them. Similarly, it would suffice if at some point in the past the verifier shared a random string with the centers. Second, we require that every voter have a discrete-log public-key in which they themselves are guaranteed to know the private key. Note that it doesn't matter if an adversary has coerced a voter to reveal this key.

Subsection 3.3 discusses how to set up a chameleon-blob system with the voters.

### 3.1 Overview

The scheme takes the following steps. We use freely the techniques of [BT94] and [CY86], adapting them to the mix-type setting.

1. For each voter $i$, the final counting center posts encrypted 1-votes and 0-votes in random order. He commits to the ordering using chameleon bit commitments. note that the voter can open these commitments arbitrarily. The center executes prove 1-0 vote to prove that he constructed the vote-pairs properly. He decommits the ordering only to the voter through an untappable secure channel.
2. Each centers shuffles the two votes for voter $i$ through the mix-net in reverse order. He commits to how he shuffled using chameleon commitments. Each execute proof SHUFFLE to prove the correctness of his action. He reveals how he shuffled only to the voter $i$ through untappable secure channel.
3. By keeping track of the initial ordering of the pair, and how they were flipped at each stage, each voter knows which vote is which. Each voter submits one of the votes sent down to him.
4. All of the voters' votes are anonymously sent to the counting center using verifiable mix-net described in Section 2. The counting center tallies the votes.

### 3.2 The Main Protocol

General Constants : $p = kq + 1$ ($p, q$ prime),
$g = (g')^k \bmod p$( where $g'$ is a generator mod$p$ )
Center $j$ Public Key : $y_j = g^{x_j} \bmod p$
Center $j$ Secret Keys : $x_j$
Voter $i$'s Public Key : $\alpha_i = g^{a_i}$
Voter $i$'s Secret Key : $a_i$
1-vote : $m_1$
0-vote : $m_0$

1. The last center $n$ executes the following with each voter $i$. He first commits a random bit string $\pi^{(i,n)}$ of length $l + 1$ to the voter using public key $\alpha_i$. For convenience let $\pi_k^{(i,n)}$ denote $k$th bit of string $\pi^{(i,n)}$. He then generates

$$v^0 = (\bar{G_n}, \bar{M_n}) = (g^{r_{2n}}, m_0 \cdot \bar{y}^{r_{2n}})$$
$$v^1 = (\bar{G_n}', \bar{M_n}') = (g^{r_{2n-1}}, m_1 \cdot \bar{y}^{r_{2n-1}})$$

where $\bar{y} = \prod y_i$. He places $(v^0, v^1)$ if $\pi_1^{(i,n)} = 0$ and $(v^1, v^0)$ otherwise. He proves that the placed pair is a combination of 1-vote and 0-vote, using prove 1-0 vote (and $\pi^{(i,n)}$) below for $l$ times, where $l$ is a suitable security parameter.
2. The counting center reveals to the voter which is 1-vote by decommitting $\pi^{(i,n)}$.

3. The next center $n-1$ executes the following with each voter $i$. He first commits a random bit string $\pi^{(i,n-1)}$ which is $l+1$ bit long to the voter. For convenience let $\pi_k^{(i,n-1)}$ denote $k$th bit of string $\pi^{(i,n-1)}$. He calculates

$$(\bar{G_{n-1}}, \bar{M_{n-1}}) = (\bar{G_n} \cdot g^{r_{2(n-1)}}, \bar{M_n} \cdot \bar{y}^{r_{2(n-1)}})$$
$$(\bar{G_{n-1}}', \bar{M_{n-1}}') = (\bar{G_n}' \cdot g^{r_{2(n-1)-1}}, \bar{M_n}' \cdot \bar{y}^{r_{2(n-1)-1}})$$

where $(\bar{G_n}, \bar{M_n})$ and $(\bar{G_n}', \bar{M_n}')$ are the votes for voter $i$ sent from the previous center. The center $n-1$ place the votes in this order if $\pi_1^{(i,n-1)} = 0$ and reverse otherwise. He proves that the placed pair is a combination of 1-vote and 0-vote, using prove SHUFFLE for $l$ times and for each interaction uses the first unused bit in $\pi^{(i,n-1)}$ for random permutation $\lambda$.
4. The center reveals to the voter how he placed by decommitting $\pi^{(i,n-1)}$.
5. Step 3-4 are repeated for center (mixer) $n-2$ down to the first center.
6. The voter, who can compute which vote of the shuffled pair is a 1-vote and which is a 0-vote, submits the desired vote to the first center of mix, which sends it down to the counting center through the mix-channel described in Section 2.
7. After the last center reveals the permuted votes, anyone can compute the number of 1-votes ($m_0$) and 0-votes ($m_1$).

**Remark:**
We need to choose $m_0$ and $m_1$ so that

$$v^0 = (g^{r_{2n}}, m_0 \cdot \bar{y}^{r_{2n}}) \text{ and}$$
$$v^1 = (g^{r_{2n-1}}, m_1 \cdot \bar{y}^{r_{2n-1}})$$

are indistinguishable. The receipt-freeness follows, as with the [BT94] protocol from the fact that the verifier can forge "proofs" that imply that the 1-vote and 0-vote were given in either order.

prove 1-0 vote

1 The prover uniformly chooses $r', r'' \in Z_{p-1}$ and calculates

$$E_0(v^0) = (g^{r'}, m_0 \cdot \bar{y}^{r'})$$
$$E_1(v^1) = (g^{r''}, m_1 \cdot \bar{y}^{r''})$$

send $E_0(v^0), E_1(v^1)$ in the order according to next unused bit in $\pi^{(i,n)}$.

2a. With probability $\frac{1}{2}$, the verifier asks the prover to reveal $r'$ and $r''$. The verifier checks if $E_0(v^0), E_1(v^1)$ is made consistently.

2b. With probability $\frac{1}{2}$, the verifier asks the prover to reveal $r_{2n} - r'$ and $r_{2n-1} - r''$. The verifier checks the following holds. $v^0$ and $v^1$ can be generated from $E_0(v^0), E_1(v^1)$.

For completeness, we summarize the chameleon bit-commitment scheme due to [BKK].

**Commitment** Sender $j$ commits 0 by $g^r$ and $\alpha_i \cdot g^r$ for 1 to receiver $i$, who knows $a_i$ satisfying $\alpha_i = g^{a_i}$

**Decommitment** Sender reveals $r$. The receiver calculates both $g^r$ and $\alpha_i \cdot g^r$ and obtain the committed bit.

**Modified Decommitment** Receiver claims he received $r - a_i$ instead of $r$.

### 3.3 Confirming the Voter Knows his Secret Key

By adopting non-interactive proofs and chameleon blobs, the voters does not need to interact with centers in a *voting booth* as was needed in [BT94] scheme, if only he can receive messages untapped. The messages must be physically untappable so that the verifier is free to lie about their contents. Also, it is necessary to make sure that voter $i$ in fact knows the discrete logarithm of $\alpha_i$ in order to have the chameleon effect. This is seemingly innocent, but must be stated as a separate assumption. The simplest solution is to assume that the verifier is given a discrete-log at some time in the distant past, such as for a public-key. We note that only one such piece of information must be given for all time. Also, while losing this discrete-log to the center is dangerous, losing it to a coercer does not affect the receipt-freeness of the protocol.

## References


[BCC88] G. Brassard, D. Chaum and C. Crépeau. *Minimum Disclosure Proofs of Knowledge.* In *JCSS*, pages 156–189. 1988.

[Ben87] J. Cohen Benaloh. *Verifiable Secret-Ballot Elections.* PhD thesis, Yale University, 1987. YALEU/DCS/TR-561.

[BKK] J. Boyar, M. Krentel and S. Kurtz. A discrete logarithm implementation of perfect zero-knowledge blobs. *Journal of Cryptology*, Vol. 2, No. 2, pp. 63–76, 1990.

[BT94] J. Cohen Benaloh and D. Tuinstra. *Receipt-Free Secret-Ballot Elections.* In *STOC 94*, pages 544–553. 1994.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM*, pages 84–88. ACM, 1981.

[CY86] J. Cohen Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *Annual Symposium on Principles of Distributed Computing*, pages 52–62, 1986.

[FOO92] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology –Auscrypt '92*, pages 244–251, 1992.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology –Crypto '86*, pages 186–199. Springer-Verlag, 1986.

[Gen94] R. Gennaro. Using non-interactive proofs to achieve independence efficiently and securely. MIT-LCS Technical Memo 515. 1994.

[NR94] V. Niemi and A. Renvall. How to prevent buying of votes in computer elections. In *ASIACRYPT '94*, pages 141-148. 1994.

[Pfi94] B. Pfitzmann. Breaking an efficient anonymous channel. In *EUROCRYPT '94*, pages 339-348. 1994.

[PIK93] C. Park, K. Itoh, and K. Kurosawa. All/nothing election scheme and anonymous channel. In *EUROCRYPT '93*, 1993.

[SK94] K. Sako and J. Kilian. Secure voting using partially compatible homomorphisms. In *Advances in Cryptology –Crypto '94*, pages 411–424. Springer-Verlag, 1994.