# Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking

Markus Jakobsson[*], Ari Juels[†] and Ronald L. Rivest[‡]

February 1, 2002

## Abstract

We propose a new technique for making mix nets robust, called *randomized partial checking* (RPC). The basic idea is that rather than providing a proof of completely correct operation, each server provides strong evidence of its correct operation by revealing a pseudo-randomly selected subset of its input/output relations.

Randomized partial checking is exceptionally efficient compared to previous proposals for providing robustness; the evidence provided at each layer is shorter than the output of that layer, and producing the evidence is easier than doing the mixing. It works with mix nets based on any encryption scheme (i.e., on public-key alone, and on hybrid schemes using public-key/symmetric-key combinations). It also works both with Chaumian mix nets where the messages are successively encrypted with each servers' key, and with mix nets based on a single public key with randomized re-encryption at each layer.

Randomized partial checking is particularly well suited for voting systems, as it ensures voter privacy and provides assurance of correct operation. Voter privacy is ensured (either probabilistically or cryptographically) with appropriate design and parameter selection. Unlike previous work, our work provides voter privacy as a global property of the mix net rather than as a property ensured by a single honest server. RPC-based mix nets also provide very high assurance of a correct election result, since a corrupt server is very likely to be caught if it attempts to tamper with even a couple of ballots.

**Keywords:** mix network, mix net, shuffle network, electronic voting, randomized partial checking, public verifiability.

## 1 Introduction

Chaum [7] introduced the notion of a *mix net* as a tool for achieving anonymity in email and in electronic elections. A mix net consists of a sequence of servers, called *mixes*. Each server receives a batch of input messages and produces as output the batch in permuted (mixed) order. Such mix nets are sometimes called *mix cascades* or *shuffle networks*. When used for voting, the input messages are the ballots of the voters. An observer should not be able to tell how the inputs correspond to the outputs; this property provides voter privacy in an electronic election. In Chaum's original proposal, before a message is sent through the mix net it is first successively encrypted with the public keys of the mixes it will traverse in reverse order; each mix then decrypts each message before sending it on to the next mix.

When a mix net is used to provide voter privacy in an election, it is desirable that it be *robust*—i.e., that

[*]RSA Laboratories, Bedford, MA 01730, mjakobsson@rsasecurity.com

[†]RSA Laboratories, Bedford, MA 01730, ajuels@rsasecurity.com

[‡]Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, rivest@mit.edu
Support provided by the Carnegie Foundation.

each mix should also output a proof that it has operated correctly. The concern is that otherwise a corrupt server could replace a ballot with another one, appropriately encrypted, without anybody noticing.

Abstractly, a robust mix net should:

1. *operate correctly*: the output should correspond to a permutation of the input,

2. *provide privacy*: an observer should not be able to determine which input element corresponds to a given output element (and vice versa) in any way better than guessing, and

3. *be robust*: provide a proof or at least strong evidence that it has operated correctly. In addition, it is beneficial if any interested party is able to check the proof or evaluate the evidence; a property called *public verifiability*.

We review previous work on robust mix nets in Section 2; numerous techniques have been proposed for achieving robust mix nets.

### 1.1 Randomized Partial Checking

We introduce a novel robustness technique, which we call *Randomized Partial Checking*, and show how it can be applied to obtain a highly efficient robust and private mix net, which we call an *RPC mix net*. We also show how an RPC mix net is well suited for use in electronic elections.

In an RPC mix net, the inputs are mixed as usual by a sequence of servers. The servers then produce *strong evidence* of their correct operation, rather than a *proof* of their correct operation. The strong evidence takes the form of a partial revelation of their input/output relation. For example, a server with $n$ inputs might reveal, for each of $n/2$ randomly selected inputs (or some other sufficiently large fraction), which is the corresponding output. (Of course, the server should have little or no control over which inputs are selected.) This procedure allows for a probabilistic verification of the correct operation of each server.

With an RPC mix net, privacy is a somewhat more delicate affair, as servers will be routinely disclosing information about their input/output relations in order to provide evidence of correct operation. We shall see how privacy can be ensured nonetheless as a global property of the RPC mix net. In one version of our proposal, adjacent servers are paired, such that if one server reveals information about a link, the paired server does not reveal information about that same link. See Figure 1 for an illustration.

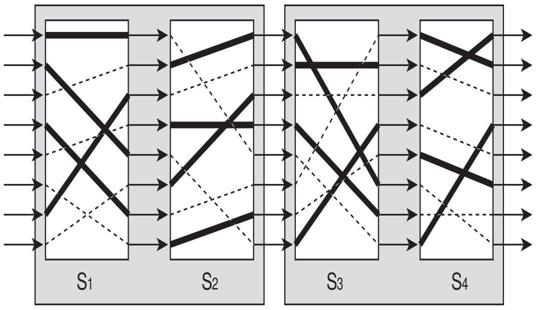


Figure 1: This figure shows a particular permutation for a mix net, partially revealed. The bold lines show input/output correspondences that are revealed; the dashed lines show correspondences that would be hidden. Server S1 is paired with S2, and server S3 is paired with S4; no input/output correspondence is revealed across a pair. Thus, to a casual observer, only the correspondences relating to the bold lines can be inferred.

Another advantage of RPC mix nets is that they are very versatile – they can be used with almost any encryption scheme, whether with or without sharing of the secret keys among the mix servers.

### 1.2 Privacy in RPC mix nets

Privacy in an RPC mix net is a different and somewhat more subtle issue than it is for a traditional mix net. In a traditional mix net, privacy is obtained whenever any one server is honest (i.e., whenever any one server keeps its input/output relation totally secret). In an RPC mix net, however, every server intentionally reveals a portion of its input/output relation. Therefore, privacy becomes a global property of the mix net rather than the result of any single honest server.

Our basic strategy for ensuring privacy is such that after the servers reveal partial information, there is still no way to connect any input with a particular output, even if some of the servers are corrupt. Using this approach, an RPC mix net guarantees privacy against any minority of cheating servers. While different privacy guarantees can be made, we consider a construction in which each element is "hidden among" at least half of all the candidate elements.

### 1.3 Robustness

Robustness of a mix net can be obtained in serveral different ways, namely cut-and-choose [17, 2]; repetition robustness [11, 12, 15]; standard zero-knowledge proofs in sorting networks [3, 13]; use of multiple participants per layer [8, 18]; error detecting techniques [14]; and techniques based on secret sharing [10, 16]. (We explain the relations between these in Section 2.)

In most of these schemes, a detected cheating attempt results in the emulation of of the cheater (such as in [14]) or the restarting of the protocol after a replacement of the cheater (such as in [17]). In some schemes, such as [8, 18], the outputs of the cheaters are simply ignored by the honest majority, and the execution continues without any interruption. (These schemes, though, tolerate a substantially lower fraction of cheaters.) In our scheme, either of the two first approaches can be taken upon detection of a cheater, although the best approach may depend on the type of encryption used. In particular, if an encryption scheme allowing re-encryption (such as ElGamal) is employed, then either approach may be taken, while emulation is the better approach in hybrid schemes, and in schemes of the Chaumian type. This is so since the operation performed by the servers on their input elements is deterministic (if we do not take the permutation aspect into consideration.) For the same reason, schemes of this latter type requires us to perform the correctness check in phase with the mixing, as opposed to after all mixing has been performed. For simplicity, we focus on schemes based on re-encryption in the following, but note that given appropriate attention to the recovery from cheating, our techniques apply straighforwardly to other types of encryption as well.

If no servers are caught cheating, it is still possible that some undetected cheating has occurred. For example, a corrupt server may have deleted one of its correct output messages and replaced it with an arbitrary incorrect one. We shall see, however, that it is very unlikely that a meaningful amount of undetected cheating has occurred, where cheating is *meaningful* if and only if it changes the outcome of the election. Thus, our solution is geared in particular towards use in election schemes or similar applications. To quantify the likelyhood that cheating occured unnoticed, we introduce the notion of *boundary checks*, and employ them to assess when the output can be relied on. In *extremely* close races, our techniques may have to be augmented by additional or alternative robustness techniques, while even in *reasonably* close races, it will suffice. For example, we show that our techniques would more than suffice to prove robustness in an election such as the recent Florida state presidential election.

### 1.4 Application to Electronic Voting

RPC mix nets are well suited to voting, since anyone can calculate strong upper bounds on the probability that an adversary could have successfully tampered with enough ballots to change the election outcome. If this probability is negligible (as it almost certainly would be in practice), the observed result of the election is endorsed as “official”. Otherwise, we may fall back to an alternative and potentially more costly scheme to count the cast votes.

Thus, our scheme is optimistic in a slightly different sense than schemes that simply assume, in the absence of detection, that there are no cheaters.

### 1.5 Outline of this paper

Section 2 reviews previous work on robust mix nets. Section 3 then provides a common framwork and common notation for discussing mix nets. Section 4 describes our main idea—that each mix server should reveal a randomly selected portion of its input/output relations. Section 5 then sketches how one can use RPC nets to implement electronic voting in a practical and trustworthy manner. Section 6 shows how RPC mix nets achieve public verifiability in the sense that any voter or other interested party can check that the probability that the election outcome is correct is extremely high.

## 2 Previous Work on Robust Mix Nets

In the first proposal for a robust mix net, due to Ogata, Kurosawa, Sako, and Takatani in 1997 [17], robustness was achieved by means of *cut-and-choose* methods. Similar techniques were later also employed in [2]. The primary drawback of this approach is its inefficiency, both in terms of computation and communication. While the schemes offer public verifiability, efficiency constraints make this feature difficult to obtain in a practical sense for large-scale elections.

An alternative technique employed by Abe [3] and similarly by Jakobsson and Juels [13] relies on more efficient zero-knowledge proofs of ciphertext equivalence. The resulting mix net construction mimics a sorting network in its architecture, but uses local random permutations instead of local sorting in its nodes. While it offers public verifiability at reasonable cost, its asymptotic behavior makes it useful primarly for batches of small or moderate sizes; it becomes impractical for large elections.

More recently, techniques developed independently by Furukawa and Sako [10] and by Neff [16] employ what may loosely be regarded as secret-sharing mechanisms to detect corruptions of data. Both of these techniques are publicly verifiable, and have costs linear in the number of inputs (and servers). While they offer features well suited for use in large-scale elections, our proposed technique achieves further efficiency and versatility.

Researchers have also considered a weakening of the requirement for public verifiability in mix nets, instead relying on a trust assumption that a certain number of servers are honest. An early technique in this vein, introduced by Jakobsson [11], is that of *repetition robustness*. Repetition robustness involves processing and comparison of several randomized instances of input items. The same technique is also employed in [12, 15]. Repetition robustness is primarily useful for very large batches.

Another approach for achieving robustness is to simply let each layer of the mixing be processed by a *set* of servers (instead of only one), basing the correctness of the result on the honesty of a majority in each layer. This technique was suggested by Desmedt and Kurosawa [8] for asymmetric ciphertexts, and later also used for hybrid encryption [18]. Diverging from the other proposals, mix nets of this type are resilient against corruption of less than a square root of the number of servers, instead of against a minority as is standard. On the other hand, the very straightforward structure makes this type of mix net trivial to analyse and understand.

While asymmetric mix networks are well suited for short plaintexts, they have problems handling longer plaintexts. These are either in the form of efficiency problems (with very large moduli) or with keeping plaintexts parts together after when passing them thorugh a mix network in a "chopped-up" manner (this, in turn, may result in lower efficiency.) Therefore, hybrid mixes have to be employed for longer plaintexts (note that these should all be of the same size after having been padded). As mentioned above, one approach, used by Abe [18], is to replicate servers. Another technique, introduced by Jakobsson and Juels [14], involves use of cryptographically-based error detection to identify cheating. This approach has the advantage of permitting symmetric and asymmetric encryption to be interleaved, leading to efficient processing of long input items. The underlying trust assumption is that a majority of servers is honest. This mix net is quite fast for a small number of inputs, although in this case is not quite as fast as [12, 15] if the inputs are short. Additionally, it only works as a decryption mix net, not a re-encryption mix net.

## 3 Notation

We now provide notation describing the operation of a simple mix net without robustness against server faults.

A voting scheme can employ either of two basic flavors of mix net.

The first of these is known as a *re-encryption mix net*. In this type of mix net, both inputs and outputs are ciphertexts under the public key of some (semantically secure) cryptosystem that admits for re-encryption without knowledge of the corresponding private key; El Gamal is a common choice. The action of each server in the net is to re-encrypt inputs and then permute them.

The second type of mix net is known as a *decryption mix net*. This is the basic mix scheme formulated by Chaum. Inputs to the mix net are ciphertexts constructed through successive encryption under the public keys of individual servers. To process inputs, each server decrypts the layer corresponding to its own public key in each ciphertext and then permutes the resulting items.

Our RPC scheme is applicable to either type of mix net. Let us now introduce general notation that is applicable to either kind of mix net.

We assume that there is a sequence of $n$ ciphertexts corresponding to input messages $M_1, M_2, \ldots, M_n$ to the mix net, each such ciphertext submitted by a distinct party $V_i$. In the application to electronic voting, $M_i$ is the ballot of voter $V_i$. These inputs are *secret*; input $M_i$ is known only to party $V_i$.

The output of the mix net is a sequence $Z_1$, $Z_2$, $\ldots$, $Z_n$. When the mix net operates correctly, this sequence is a permutation of the input sequence.

We assume that there are one or more *public parameters* (e.g., public keys) of the mix net, denoted collectively as $\mathbf{PK}$, known to all voters. There are also one or more *secret parameters* (e.g., secret keys), denoted collectively as $\mathbf{SK}$, which may be shared among the servers, or alternatively by some other set of authorities.

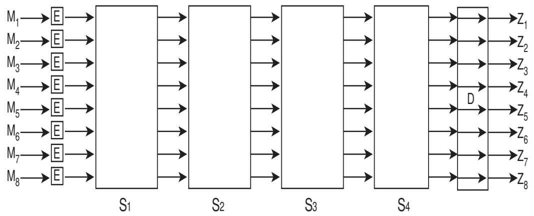


Figure 2: Generalized mix net, shown for n=8, t=4. The $n$ inputs $M_1$, $M_2$,..., $M_n$ are first privately encrypted by their providers using encryption function $E$. The $t$ mix servers $S_1$, $S_2$, $\ldots$, $S_t$ then each privately transform and permute their inputs, and provide the result to the next server. The final decryption operation $D$ yields a permuted version $Z_1$,$Z_2$, $\ldots$, $Z_n$ of the original input sequence. This final stage may be integrated in the previous transforms.

The general operation of a mix net is depicted in Figure 2. There is an initial encryption of each input message by its provider. The resulting sequence of ciphertexts is then provided to the first mix server $S_1$ of a sequence of $t$ mix servers $S_1$, $S_2$, $\ldots$, $S_t$. Each mix server cryptographically transforms each input, permutes the results, and provides the result as input to the next server. A final decryption operation produces the sequence $Z_1$, $Z_2$, $\ldots$, $Z_n$ which is a permutation of the original input sequence of messages.

We assume the existence of a public *bulletin board* where messages (digitally signed by their poster) can be posted by anyone, and read by anyone. This board is written in *append-only* mode; nothing can be deleted or modified once posted. The original encrypted input sequence to the first mix server, the output sequence of each mix server, and the final decrypted message sequence will all be posted on the bulletin board.

We denote the initial encrypted version of message $M_i$ as $C_{i,0}$. That is,

$$C_{i,0} = E_{\mathbf{PK}}(M_i) \; .$$

The sequence $C_{1,0}$, $C_{2,0}$, $\ldots$, $C_{n,0}$ is input to the first server.

The encryption function $E$ must be *non-malleable* or *plaintext aware* [9, 4, 6]. Thus, it may consist of a ciphertext in an underlying cryptosystem such as El Gamal, coupled with a proof of knowledge of the corresponding plaintext [19, 11]. The reason for this is to prevent attacks in which one (potentially corrupt) voter posts a re-encryption of the ballot of some other voter.[1]

Server $S_j$, for $1 \leq j \leq t$, cryptographically *transforms* each input $C_{i,j-1}$ using a cryptographic transformation function $X_j$. Here $X_j$ may depend on secret key information $SK_j$ known only to server $S_j$, as well as on the public parameters $\mathbf{PK}$. The transformation $X_j$ may also be randomized. Each server $S_j$ also permutes its inputs based on a secret permutation $\pi_j$ of $\{1, 2, \ldots, n\}$, so that

$$C_{i,j} = X_j(C_{\pi_j(i),j-1}) \; . \qquad (1)$$

In the case of a re-encryption mix, a final decryption operation $D$ may be applied to the output of the final mix server:

$$Z_i = D_{\mathbf{SK}}(C_{i,t}) \; .$$

[1] For example, suppose that a corrupt voter suspects another, target voter of having submitted an unusual write-in vote like "Julius Caesar". The corrupt voter could re-encrypt and re-post the vote of the target voter. If "Julius Caesar" appears twice in the finally tally, then the suspicions of the corrupt voter would be confirmed. Similar attacks can also, as is well known, be employed for vote buying or coercion.

This decryption operation will be null in the case of a decryption mix net, since the $X_j$ transforms performed all necessary decryptions. In the case of a re-encryption mix net, one or more *decryption authorities* knowing $\mathbf{SK}$ perform this final decryption.

For a Chaumian mix net (i.e. a decryption mix net), the public keying material $\mathbf{PK}$ includes an individual public key $PK_j$ for each server $S_j$ in the underlying cryptosystem, e.g., RSA. Server $S_j$ then holds one of the corresponding private keys, $SK_j$. Thus, the encryption scheme $E$ in this case involves successive (random-padded) encryption of the message $M_i$ under $PK_t, PK_{t-1}, \ldots, PK_1$ respectively. To satisfy the need for plaintext awareness in $E$, we might employ an encryption scheme like OAEP-based RSA [5]. In a decryption mix net, we naturally replace $X_j$ with a *decryption* function: each server $S_j$ decrypts a ciphertext $C_{\pi_j(i),j-1}$ using its private key $\mathbf{SK_j}$, thereby stripping away a ciphertext layer. As the output of server $S_t$ is thus a set of plaintexts, there is no need in a Chaumian mix net a further decryption operation $D$.

For a re-encryption mix net, the initial encryption function may be a suitable plaintext-aware version of El Gamal, as noted above. (Note that the corresponding proofs of knowledge do not have to be passed through the mix network, but stripped off after having been checked initially.) Each cryptographic transform $X_j$ will be a randomized re-encryption. The final decryption operator $D$ will be El Gamal decryption.

### 3.1 Committing to private permutations

To assist in verification of correct behavior as explored in the next section, each server $S_j$ supplements its list of output values with a commitment to its private permutation $\pi_j$. So as to enable partial revelation of $\pi_j$, servers in fact commit to individual input/output mappings, as we now describe.

To provide rough notation, let $\zeta_w[i]$ denote a commitment to integer $i$ under witness $w$. There are two equivalent ways for a server $S_j$ to commit to its private permutation $\pi_j$. The first is to express $\pi_j$ in terms of mappings of input elements to output elements, i.e., as a list of commitments to the sequence $\{\pi_j(1), \pi_j(2), \ldots, \pi_j(n)\}$. We denote a commitment of this form by $\Gamma_j^{(In)} = \{\zeta_{w_{ji}}[\pi_j(i)]\}_{i=1}^n$. A second way to specify the private permutation $\pi_j$ is in terms of the mappings of output elements to input elements, i.e., as a commitment to the sequence $\{\pi_j^{-1}(1), \pi_j^{-1}(2), \ldots, \pi_j^{-1}(n)\}$. We denote the commitment to this list by $\Gamma_j^{(Out)} = \{\zeta_{w_{ji}}[\pi_j^{-1}(i)]\}_{i=1}^n$. For either of the two forms of commitment, we let $\gamma_{i,j}$ denote the $i^{th}$ commitment of server $S_j$.

In our constructions described in the next section, a server $S_j$ will provide with its output a commitment to $\pi_j$. The server will employ the form $\Gamma_j^{(In)}$ or $\Gamma_j^{(Out)}$ depending on its role in the mix network.

In practice, in the interest of speed, we might instantiate the commitment scheme $\zeta$ by means of a hash function $h$ such as SHA-1. To commit to an integer $i$, the committer selects a random bitstring $w$, and computes $\zeta_w[i] = h(w \parallel i)$, where $\parallel$ denotes bitstring concatenation.[2] It may be observed that this form of commitment is computationally binding, with security dependent on the collision-freeness of $h$. Provided that $w$ is long enough, the commitment is unconditionally hiding with high probability over the choice of witness. This is because for a given image $h(w \parallel i)$ there are likely to correspond many values of $w'$ and $i$ such that $w' \parallel i'$ constitutes a valid preimage.

# 4 Randomized Partial Checking of a Mix Net

Of course, anyone may check that each server has produced the same number of outputs as it has inputs. But a server might have deleted a proper output, and replaced it by a copy of another one, or by an output that it generated itself. In this latter case, it would be an appropriately encrypted output.

In our proposal, each server will – during the checking phase – reveal a fraction $p > 0$ of its input/output correspondences. The subset to be revealed is selected by the other servers, or by using a random oracle. Thus, only some messages will have their origins hidden by the first mix server. But as the messages progress through the net, eventually every message will have its origin hidden. For an electronic election, voter privacy then emerges as a global property of the mix net, not a local property of each mix server.

In our formulation of the problem, the penalty for misbehavior by a server will be very large. We thus presume that the threat of detection of misbehavior by a server will be enough to ensure that the server will behave properly. We do not worry about the possibility that some server will try to block an election by, say, refusing to carry out its duties. (Threshold mix nets are designed to counter this threat; another approach would be to require that each server escrow shares of its secret key with the other servers before voting begins.)

Similarly, the chance that a server who attempts to substitute ballots will be caught will go up exponentially fast with the number of ballots he attempts to replace. Thus, a server could not reasonably expect to get away with changing more than a single ballot, or possibly two. But even when tampering with a single ballot, his chance of discovery is more than one-half, for reasonable settings of the system parameters, and so we presume that he will be deterred from even attempting to cheat.

## 4.1 Revealing a particular input/output correspondence

In the verification stages of our protocol, a server is asked to reveal a collection of input/output correspondences. If the server has committed to in-

[2] To ensure input of a 512-bits block for the compression function in the case where $h$ is chosen to be, e.g., SHA-1, it is convenient to express the integer $i$ as a string of $\lceil \log_2 n \rceil$ bits, and $w$ as a bitstring of length $512 - \lceil \log_2 n \rceil$.

put mappings, these correspondences are specified in terms of the ordering of inputs to the server. Otherwise, they are specified in terms of outputs to the server.

Suppose that server $S_j$ wishes to reveal information allowing anyone to verify a particular input/output correspondence. Let us suppose that input $C_{k,j-1}$ maps to $C_{i,j}$. That is, the secret permutation $\pi_j$ known only to $S_j$ maps $i$ to $k$ (see equation (1)).

The server reveals the triple

$$(k, i, R_{jki}),$$

where $R_{jki}$ is the information necessary to validate equation (1). For a decryption mix net, this information $R_{jki}$ make take the form of random padding created by the initial provider and used when encrypting $C_{i,j}$ to obtain $C_{k,j-1}$. For a re-encryption mix net, this information $R_{jki}$ takes the form of random parameters used to control the re-encryption, or a proof of knowledge of these.

Server $S_j$ additionally reveals its commitment to the mapping from $C_{k,j-1}$ maps to $C_{i,j}$. That is, if it provided a commitment to $\pi_j$ of the form $\Gamma_j^{(In)}$, then it decommits $\gamma_k$, i.e., its commitment to $\pi_j(k)$. If the server provided a commitment $\Gamma_j^{(Out)}$, then it decommits $\gamma_i$, i.e., its commitment to $\pi_j^{-1}(i)$.

## 4.2 Determining which correspondences to reveal

Clearly, a server should not know which input/output correspondences it will have to reveal until after it has committed its output sequence of ciphertexts to the bulletin board.

We first focus on the problem of having a random seed committed to before server $S_j$ produces its output. This seed will then help determine which input/output correspondences server $S_j$ should reveal. There are a variety of ways of achieving this goal; we suggest the following straightforward approach.

After the close of the election and prior to the opening of input/output relations, servers jointly compute a random seed $R$. They may accomplish this by having every server $S_j$ publish a commitment to a value $R_j$ selected uniformly at random from some appropriate set. All servers then decommit and compute $R$ as a combination of the $R_j$ values; for example, they might compute $R = \oplus_{j=1}^{t} R_j$.

Let $\mathcal{BB}$ here denote the full contents of the bulletin board after all servers have published their full transcripts, i.e., all inputs, outputs, and commitments (but for the moment excluding input/output relations). Note that new public transcripts are constantly added to the bulletin board during the mix process: thus, $\mathcal{BB}$ denotes both the bulletin board and this *dynamically changing* value. Servers combine the random value $R$ with $\mathcal{BB}$ through use of an appropriate hash function $h$, computing a random value $Q = h(R, \mathcal{BB})$. The purpose of incorporating $\mathcal{BB}$ into the random seed $Q$ in this manner is to achieve public verifiability for the mix scheme, as we discuss later.

For each server $S_j$, a seed $Q_j$ derived from $Q$ can be used to determine what challenges the server needs to answer. Here, $Q_j$ may be computed straightforwardly using an appropriate hash function $h$. We might, for example, compute $Q_j = h(Q, j)$.

We next assume the existence of two predicates $P_{In}$ and $P_{Out}$ that determine which inputs and which outputs should have their input/output correspondences revealed. More precisely:

- If $P_{In}(Q_j, k)$ is true, then server $S_j$ must reveal the input/output pair containing $C_{k,j-1}$ as input.
- If $P_{Out}(Q_j, i)$ is true, then server $S_j$ must reveal the input/output pair containing $C_{i,j}$ as output.

(A correspondence may be revealed because either because $P_{In}$ specifies it, or because $P_{Out}$ specifies it.) Any other input/output correspondences should *not* be revealed. These predicates may also depend on other global parameters. For example, there may be a global selection probability $p$ that is intended to specify the fraction of correspondences to be revealed. For some versions of our scheme it may be that $P_{In}$ is always false, or that $P_{Out}$ is always false. (That is, the pairs to be opened may be entirely specified by their input positions, or by their output positions.)

We next present two variations on the details; the second scheme is the one we favor.

### 4.3 Scheme One – Independent Random Selections

In this scheme, server $S_j$ furnishes a commitment $\Gamma_j^{(Out)}$ on mappings from outputs to inputs. When input/output relations are revealed here, $P_{In}$ is always false, and $P_{Out}$ is true with probability $p$. (Imagine, say, $p = 1/2$.) For example, we might have $P_{Out}(Q_j, i)$ true whenever the low-order bit of $h(Q_j, i)$ is one, for a specified pseudo-random hash function $h$.

When $t$ is large enough, with high probability every path from an initial input $C_{k,0}$ to the corresponding final output $C_{i,t}$ will be "broken" (contain some unrevealed link).

For $p = 1/2$, if

$$t \geq \log_2 \left(\frac{n}{\epsilon}\right)$$

then the chance that there exists some final output that can be linked to its initial input is less than $\epsilon$.

We note that if a final output can not be linked to its initial input, then there are at least $n/2$ inputs from which it could have been produced. Thus, the ambiguity of the input corresponding to a given output may extend over $n/2$ elements, rather than the full $n$ elements. For many practical applications, such as voting, this should be acceptable.

This scheme works fine, but takes more rounds (a larger value of $t$) than we would prefer. For example, with $n = 4096$ and $\epsilon = 2^{-24}$ we need $t \geq 36$ rounds. It might in practice be necessary to use the available servers in some sort of "round-robin" fashion to achieve the necessary number of rounds.

### 4.4 Scheme Two – Pairwise Dependent Selections

In scheme two, adjacent servers are "paired", letting each server be a member of exactly one such pair (see Figure 1). In particular, we assume an even number $t$ of servers, and regard each pair of adjacent odd and even-numbered servers as a cohesive unit. When servers reveal input/output relations, the two servers in a pair each reveal non-overlapping sets of such relations. For simplicity of analysis, we assume $p = 1/2$ here. This is to say that each server in a pair reveals half of its input/output pairs on average, and the other server reveals the complementary half, i.e., the relations not revealed by its twin. (Of course, neither server would make its revalations until both of them have committed to their outputs.)

In this scheme, each odd-numbered server $S_j$ publishes a commitment $\Gamma_j^{(In)}$ on the mapping from input elements to output elements; conversely, each even-numbered server $S_j$ publishes a commitment $\Gamma_j^{(Out)}$ on the mapping from output elements to input elements.

Let us now specify the process for revealing input/output relations. Suppose that $(S_j, S_{j+1})$ is a server pair, where $j$ is odd. Then

- $P_{In}(Q_j, k)$ is always false.
- $P_{Out}(Q_j, i)$ is true with probability $1/2$.
- $P_{In}(Q_{j+1}, i)$ is true if and only if $P_{Out}(Q_j, i)$ is false.
- $P_{Out}(Q_{j+1}, l)$ is always false.

The privacy guarantee of this variant is based on a simple observation: Provided that a (passive) adversary controls only a minority of the servers, there is at least one server pair that is entirely honest. Thus, suppose that the adversary is given complete side information regarding all input/output correspondences for all servers other than this honest pair. Then in the view of the adversary, every voter input is mixed uniformly with a known half of the other inputs. It follows that for any input value, the adversary[3] can at best identify the corresponding output value with probability $2/n$. This holds no matter how many servers there are, i.e., irrespective of $t$, so long as at least $t/2 + 1$ servers are honest.

In the context of an election, this privacy guarantee is quite satisfactory from a practical perspective. Stated loosely, it specifies that any ballot is hidden among those of half of the electorate. Provided we are willing to accept this guarantee, rather than full hiding, this proposal presents attractively practical functionality.

## 5 Electronic Voting Based on RPC mix nets

We are now ready to sketch a simple election scheme using an RPC mix net.

**System Setup.** Herein, the authorities select mix servers, publish the public keys of these, certify voters, and distribute appropriate protocols, which are assumed to be certified and correct.

**Ballot Preparation and Encryption.** Each voter $V_i$ prepares his plaintext ballot $B_i$. He then computes a ciphertext $C_{i,0} = E_{\mathbf{PK}}(B_i)$. Voter $V_i$ signs $C_{i,0}$ with his own private signing key and posts it to the bulletin board.

Each voter prepares his or her ballot by encrypting the value that encodes the ballot using the public key(s) of the authorities. This may be done by sequential encrypting using the public keys of the participating servers, starting with the last one in the mix net – here, the encryption may either be a plain asymmetric encryption, or a hybrid encryption. We refer to [14] for a description of hybrid encryption techniques. Alternatively, the encryption may be performed using the public key of the authorities. As noted earlier, the encryption technique used should be "plaintext-aware."

[3] This assumes ideal cipher characteristics. Under normal computational hardness assumptions on the underlying cipher, the adversary has some additional, negligible advantage.

**Initial Ballot Checking.** When the balloting phase is closed, all servers check the validity of the posted ciphertexts, elimating by consensus any ciphertexts that are ill-formed. They also eliminate any duplicate ciphertexts (preserving only the first posted copy). Without loss of generality, we let this result in $n$ well-formed ciphertexts.

**Permutation Commitment.** Each server $S_j$ selects a permutation $\pi_j$ on $n$ elements uniformly at random. The server publishes to the bulletin board a commitment to $\pi_j$, either $\Gamma_j^{(In)}$ or $\Gamma_j^{(Out)}$ (depending on our choice of mix variant and the parity of $j$).

**Mix Net Processing.** At this point, each server $S_j$ in turn accepts $n$ input ciphertexts $\{C_{i,j}\}_{j=0}^{t-1}$. The server applies $X$ to each of them, permutes the resulting ciphertexts according to $\pi_j$, and outputs the result to the bulletin board, along with a digital signature thereon.

**Correctness Check.** The operation of the mix servers is verified as previously outlined.

If any server is found to have cheated, and the mixing is based on re-encryption, then the corrupted server is either emulated or replaced. In the latter case, the protocol is restarted at the beginning of the mixing stage; in the former at the stage of the emulated server. If the mixing is based on decryption, then the cheater is emulated.

If re-encryption mixing is used, then the outputs of the last mix server are decrypted at the end of the correctness check, assuming this succeeds. The decryption typically would be performed by a quorum of servers of the authority sharing its secret key. (Note that these may be different from the mix servers as long as they collectively trust that a sufficient number of mix servers were honest.) Each decryption would be associated with a publicly verifiable proof of correct decryption (which typically means a proof of correct exponentiation.)

**Ballot Decryption.** Once the mixing operation is complete, the holders of $\mathbf{SK}$ (the mix servers or some other entities) jointly decrypt all output ciphertexts, yielding the full list of plaintext ballots, if applicable.

**Boundary Check.** The authorities determine the minimum number of ballots that would have to change in order to alter the outcome of the election, given the tally output at the end of the correctness check stage. They then compute the probability that this number of ballots could have been altered by cheaters, without these being detected.

In particular, suppose that alteration of at least $\kappa$ ballots would have been necessary to affect the election outcome. That is, $\kappa$ is one-half the difference in vote count between the winner and the runner-up, rounded up. The authorities estimate the probability that an adversary could have manipulated $\kappa$ ballots without detection. (We give a bound on this probability for our proposal below.)

If this estimate represents an acceptable failure probability (which we expect to be almost always the case), then the mix servers proceed to the endorsement phase; otherwise they invoke an alternative mix net on the same inputs with a stronger guarantee of correctness.

**Endorsement.** If both the correctness check and the boundary check succeeds, then the output is considered valid. The values needed for publicly performing the correctness check are published along with the final tally. (The initial contents of the bulletin board are assumed to already be public.) Everybody can perform the verifications of the correctness check (including the potential decryption verifications at its end); and then verify that the boundary conditions are satisfied.

### 5.1 Boundary probability

To compute boundary probabilities for our scheme, let us consider a *centralized* adversary, i.e., one that is capable of coordinating (in a static manner) the actions of a minority of servers and an arbitrary number of voters. All other servers and voters are assumed to be honest. Given no evidence of cheating, the question we aim to answer is this: What is the probability that the adversary could have altered votes in such a way that the apparent election outcome is not the correct one? For simplicity of presentation, we focus our analysis here on our second protocol variant involving "paired" servers, and assume a re-encryption mix with correct decryption of output ciphertexts. As a further simplifying assumption, we regard the underlying cipher and commitment schemes as "ideal", i.e., as providing information theoretic security. For $p = 1/2$, we make the following claim.

**Claim 1** *Suppose that the adversary alters elements in the mix such that the observed election tally differs by $\kappa$ votes from the correct one. Then the probability that the adversary goes undetected is at most* $1/2^{\kappa}$.

**Proof:** (sketch) Now let us first consider a server $S_j$ such that $j$ is odd, i.e., the first server in a pair. For such a server, let us define the *antecedent* of an output ciphertext $C_{k,j}$ to be an input ciphertext $C_{i,j-1}$ with the following properties: (1) $C_{k,j}$ represents a valid re-encryption of $C_{i,j-1}$ and $\gamma_i$ is a commitment to the value $\pi_j(i) = k$. Observe that $S_j$ cannot successfully open the input/output relationship for a given output ciphertext without a correct antecedent.

Now consider a server $S_j$ such that $j$ is even, i.e., the second server in a pair. For such a server, let us define the *successor* of an input ciphertext $C_{i,j-1}$ to be an output ciphertext $C_{k,j}$ with the following properties: (1) $C_{k,j}$ represents a valid re-encryption of $C_{i,j-1}$ and $\gamma_k$ is a commitment to the value $\pi_j^{-1}(k) = i$. Observe that $S_j$ cannot successfully open the input/output relationship for an input without a correct successor.

We refer to a ciphertext that lacks a correct antecedent or lacks a correct successor as a *dud*. Based on our definitions of antecedents and successors, a dud must be an "intermediate" ciphertext, i.e., the output of an odd-numbered server or, equivalently, the input of an even-numbered one. A given dud will be detected with probability at least 1/2, as either its antecedent or successor must be checked. It may also be seen in our scheme that duds are checked independently, i.e., as independent events.

Let us consider "paired" servers $(S_j, S_{j+1})$. Suppose that the input and output ciphertexts to this pair of servers differ in at least $K_{j,j+1}$ values. More precisely, suppose that any one-to-one mapping $f$ from outputs to inputs excludes at least $K_{j,j+1}$ output elements. It may be seen there exists such a one-to-one mapping $f$ that includes at least one distinct input/output pair of ciphertexts for every intermediate ciphertext that is not a dud. Therefore, there are at least $K_{j,j+1}$ duds among the intermediate ciphertexts. It is clear that $K \leq K_{1,2} + K_{3,4} + \ldots + K_{n-1,n}$. (Intuitively, the total number of altered ciphertexts at each server pair cannot exceed the number of ciphertexts altered across the entire mix network.) Therefore, there are at least $\kappa$ duds among the intermediate ciphertexts published by all server pairs. Since each dud is detected independently with probability at least 1/2, the claim follows. □

**Example:** Given an output tally of 46 Republican votes and 54 Democratic votes in a small election, authorities would conclude that in the worst case, an attacker might have swung the election through manipulation of a minimum of four initially Republican votes. (This would be possible, for example, if the true tally were 50 Democratic vs. 50 Republican, for example.) By Claim 1, the probability that an adversary might have swung the election is at most $1/16$.

**Example:** While the correctness assurance in the above example is very low, a more realistic example gives a substantially lower adversarial success probability. Let us consider the recent U.S. Presidential election in Florida which yielded a tally with some 2,910,074 votes for Bush and some 2,909,114 votes for Gore [1]. For these tallies to be produced from ballots in which there was an exact tie or in which Gore obtained more votes, a minimum of 480 votes would have to be manipulated. By Claim 1, the probability of this would be at most $2^{-480}$, which is truly negligible and far smaller than the probability of breaking a typically parameterized crypto system.

In typical circumstances, Claim 1 represents an overestimate of the success probability of such an attacker. In particular, our computation here assumes that the attacker alters ballots in the optimal way. This is possible for an adversary corrupting the first few servers if voters register with their parties – otherwise, the adversary could only guess what ballots to alter.

## 6 Public Verifiability

To define the property of public verifiability in a mix network, we require a stronger adversarial model than for our definitions of privacy or correctness. In particular, we must assume an adversary that potentially controls *all* servers and *all* voters. This is an unrealistically pessimistic assumption, but aims to characterize the security of the mix scheme in the worst case.

In defining public verifiability, we consider a verification function, which we denote by ver, that is efficiently computable by any entity, whether or not the entity participates in the mixing process. Input to ver includes the contents of the bulletin board at the conclusion of the mixing process; in particular, it includes the set of ciphertexts input to the mix network $C_{In} = \{C_{0,i}\}_{i=1}^{n}$, the set of output ciphertexts $C_{Out} = \{C_{t,i}\}_{i=1}^{n}$, and all commitments, input/output relationships, and other evidence published by all servers. The function ver outputs "correct" if the output of the mix network is a correct representation of the input, or appears to be such; it outputs "incorrect" otherwise.

The standard definition of public verifiability states, loosely speaking, that a mix network is publicly verifiable if for some verification function ver, the adversary cannot feasibly produce input that falsely yields the output "correct". In other words, an adversary should not be able to spoof a verification function ver into accepting a "mismatched" pair $(C_{In}, C_{Out})$, i.e., a pair such that the set of plaintexts corresponding to $C_{Out}$ is not equal to that corresponding to $C_{In}$.

Our scheme achieves a somewhat weaker version of public verifiability. An adversary with full control of all players in our scheme can, strictly speaking, cause (with some probability) a verification function ver to accept a mismatched pair $(C_{In}, C_{Out})$. What such an adversary *cannot* do, however, is create a sizable discrepancy between inputs and outputs to the mix network. More precisely, we show that in order to alter $\kappa$ posted votes in an election scheme with high probability, the adversary must perform computational effort roughly $2^{\kappa}$. In consequence, our scheme provides public assurance that no adversary could have feasibly altered, say, 160 votes in the election. (Furthermore, we know that it is infeasible to modify even a much smaller number of votes if not all mix servers collude – this provides further reassurance of the correctness in case of narrow margins.)

Recall that all the servers have to commit to their permutations, as well as to their portion of what determines the challenges. This efficiently makes the protocol deterministic after it has begun, and makes it impossible for a colluding set of servers to select permutatations so that only "clean" elements will be verified.

Furthermore, recall that servers reveal input/output relations according to a random seed $Q$. This seed is computed by applying a hash function $h$ to the contents of the bulletin board after all mixing has taken place (but prior to verification, of course). Modeling $h$ as a random oracle, we may assume that for every attempt on the part of the adversary to produce a transcript that spoofs the verification function, the adversary must make an oracle call to determine what challenges the servers respond to. We consider a verification function ver that checks all revealed input/output relations in the obvious manner. Given this model, and assuming $p = 1/2$, we make the following claim.

**Claim 2** *Suppose that an adversary with full control of all servers and voters wishes to generate a pair $(C_{In}, C_{Out})$ and bulletin board contents, i.e., server transcripts, with the following property. The set of plaintexts corresponding to $C_{In}$ differs from that corresponding to $C_{Out}$ by at least $\kappa$, but* ver *outputs "correct". With $q$ queries to the oracle, the adversary will be successful with probability at most $q2^{-\kappa}$, for a number of queries $q$ to the random oracle.* □

Given this claim, we may state as a rough rule of thumb that the results of an election are publicly verifiable in a meaningful way if the winner leads by a margin of at least 160 votes. In this case, an adversary that performs computation $2^{80}$ (more precisely, makes $2^{80}$ oracle calls) has success probability at most $2^{-80}$. In a practical setting, however, this security analysis is rather conservative. It may be relaxed somewhat under assumptions such as the following.

1. *Many voters are honest:* If a voter does not collaborate with the adversary, then her ciphertext randomizes $Q$ in a manner previously unpredictable to the adversary. In consequence, the adversary can only make useful oracle queries during the interval of time between the last vote posted by an honest voter

and the time that the tally is output. This places a practical restriction on the amount of computing power the adversary can bring to bear on manipulation of the election since it forces the adversary to commence the attack after the "honest vote(s)" have been colleted, and thereby prevents a "pre-computation attack."

2. *The election includes many ballots:* A second practical security against attacks is obtained by forcing the recomputatation of long hashes in order to succeed with an attack. Namely, recall that the full contents of the bulletin board must be hashed using $h$ in order to compute the seed $Q$. In a large election, therefore, an oracle query is an expensive operation. This restricts (by some medium-sized factor) the number of oracle queries an adversary with a given amount of computing power can make.

Of course, if the tally yielded by our scheme involves too small a margin of victory to ensure public verifiability, it is always possible to apply a different and more expensive, but publicly verifiable mix network to the posted votes, e.g., [10, 16].

## 7 Discussion

The most significant advantage of the mix scheme we propose here is that the proofs are exceptionally simple; they merely involve revealing the randomizing factors for the randomized encryption operations. No zero-knowledge proofs are required. The scheme is therefore exceptionally efficient.

With use of a Chaumian mix net, the ballots may have arbitrary size or content. We may have write-in votes, or large ballots, without difficulty.

An adversary controlling some minority group of servers may try to replace $\kappa$ ballots with its own substitutes. Given $p = 1/2$, the chance of the adversary succeeding without detection is at most $1/2^{\kappa}$. Thus trying to cheat by more than a ballot or two is risky. A cheating server must either confess to cheating or (equivalently) fail to produce a required proof. The penalties for cheating would be so severe as to preclude the attempt.

In summary, we believe that RPC mix nets are an interesting and practical approach for obtaining voter anonymity in an electronic voting system.

## References


[1] George W. Bush, et al., petitioners v. Albert Gore, Jr., et al., No. 00-949, Supreme Court of the United States, 531 U.S. December 12, 2000.

[2] M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In K. Nyberg, editor, *EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 437–447. Springer-Verlag, 1998.

[3] M. Abe. Mix-networks on permutation networks. In K-Y. Lam, E. Okamoto, and C. Xing, editors, *ASIACRYPT '99*, volume 1716 of *Lecture Notes in Computer Science*, pages 258–273. Springer-Verlag, 1999.

[4] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology - CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.

[5] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In R. Rueppel, editor, *Advances in Cryptology–Eurocrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.

[6] M. Bellare and A. Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In M. Wiener, editor, *Advances in Cryptology - Proc. Crypto 99*, volume 1666

of *Lecture Notes in Computer Science*, pages 519–536. Springer-Verlag, 1999.

[7] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[8] Y. Desmedt and K. Kurosawa. How to break a practical mix and design a new one. In B. Preneel, editor, *EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 557–572. Springer-Verlag, 2000.

[9] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *Proceedings 23rd ACM STOC*, pages 542–552, 1991.

[10] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In J. Kilian, editor, *CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer-Verlag, 2001.

[11] M. Jakobsson. A practical mix. In K. Nyberg, editor, *EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 448–461. Springer-Verlag, 1998.

[12] M. Jakobsson. Flash mixing. In *PODC '99*, pages 83–89. ACM, 1999.

[13] M. Jakobsson and A. Juels. Millimix: Mixing in small batches, June 1999. DIMACS Technical Report 99-33.

[14] M. Jakobsson and A. Juels. An optimally robust hybrid mix network. In *PODC '01*, 2001.

[15] M. Mitomo and K. Kurosawa. Attack for flash MIX. In T. Okamoto, editor, *ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 192–204. Springer-Verlag, 2000.

[16] A. Neff. A verifiable secret shuffle and its application to e-voting. In P. Samarati, editor, *ACM CCS '01*, pages 116–125. ACM Press, 2001.

[17] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *Proc. ICICS '97*, volume 1334 of *Lecture Notes in Computer Science*, pages 440–444, 1997.

[18] M. Ohkubo and M. Abe. A length-invariant hybrid mix. In T. Okamoto, editor, *ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 178–191. Springer-Verlag, 2000.

[19] Y. Tsiounis and M. Yung. On the security of ElGamal-based encryption. In *Workshop on Practice and Theory in Public Key Cryptography (PKC '98)*. Springer, 1998.