Auditable Privacy: On Tamper-evident Mix Networks

Jong Youl Choi[1], Philippe Golle[2], and Markus Jakobsson[3]

[1] Dept. of Computer Science, Indiana University at Bloomington, IN 47405, USA
jychoi@cs.indiana.edu
[2] Palo Alto Research Center, 3333 Coyote Hill Rd, Palo Alto, CA 94304, USA
Philippe.Golle@parc.com
[3] School of Informatics, Indiana University at Bloomington, IN 47406, USA
markus@indiana.edu

Abstract. We introduce the notion of tamper-evidence for mix networks in order to defend against attacks aimed at covertly leaking secret information held by corrupted mix servers. This is achieved by letting observers (which need not be trusted) verify the absence of covert channels by means of techniques we introduce herein. Our tamper-evident mix network is a type of re-encryption mixnet in which a server proves that the permutation and re-encryption factors that it uses are correctly derived from a random seed to which the server is committed.

Keywords. Mix network, covert channel, malware, observer, subliminal channel, tamper-evident.

1 Introduction

In several countries, experiments with electronic voting are taking place. While the primary political goal is to increase voter turnout by allowing for streamlined casting of votes, electronic voting also offers substantial benefits in terms of precision, speed of tallying and privacy guarantees. The flip-side is the difficulty to guarantee these properties, and maintain security when under attack. Electronic voting, not surprisingly, has been at the heart of intense debate.

In electronic voting, just as in manual voting, security properties related to concrete phenomena are easier to guarantee than those related to abstract phenomena. In particular, the desirable property of correctness (the accurate counting of votes) is easier to guarantee than privacy (retaining secrecy of who voted for whom). Tallying – while time consuming and subjective in its current incarnation – is by nature easily auditable. One can duplicate functionality, and count votes in multiple ways in order to ascertain that each vote was counted exactly once. However, no similar auditing process has been proposed to verify that privacy was maintained – neither for manual nor for electronic voting. The reason, informally stated, is that *a leak is a leak is a leak.*

Failure to guarantee privacy is particularly severe in the social context of vote buying, and the technical context of malicious code, and makes any transition to electronic elections fraught with the risk of large-scale abuse. In particular, it was shown [24, 25] how covert channels can be employed to intentionally (and unnoticeably) leak secret information to collaborators. Covert channels allow tallying machines to leak either their secret keys, the state of their pseudo-random generators, or information about the votes they process. There is also real-life evidence [21] that malicious code (written to specification) has been used to spy on voters. While public code audits may address such concerns to some extent, they are hardly a panacea, especially given the difficulty of ascertaining that the audited code is in fact the code that gets loaded and deployed.

We study how to ensure public verifiability of privacy for synchronous mix networks, with direct applications to electronic elections (see, e.g., [2]). We consider an adversary that *fully controls* all servers in a mix network at all time, *except* during an initial setup phase. In the setup phase, the servers are free from adversarial control and can establish and exchange keys. The adversary only gains control over the servers after the completion of the setup phase. This models both typical malware attacks, and attacks in which the software developer writes software that "switches behavior" [28] to a malicious mode after some initial testing has established that the software is correct. We note that corruption during the setup phase can be detected using zero-knowledge proof techniques such as, for example, Juels and Guajardo's scheme [9].

Following [3], we assume the existence of *observers*, whose sole purpose is to monitor the input-output behavior of servers being observed, and determine if any of the generated transcripts could contain information that should not have been included. Observers are not provided with any secret information. Consequently, we do not have to trust observers – apart from having to trust that at least one of them is honest. There is no limit on the number of possible observers, and there is no way for the adversary to determine how many there are. Moreover, given that we employ *undercover* observers, i.e., observers that do not need to interact with any mix server in order to perform their duties (except for when they raise alerts), there is also no clear way to locate them.

While [3] is concerned with the potential leak of private key information from corrupted servers of a certification authority, we focus instead on *any* type of leak from mix servers. (In particular, we consider both leaks of the secret key and leaks that somehow reveal parts of the permutation applied by the mix network.) Although the main principles are closely related on a conceptual level, the technical approaches differ in more ways than they coincide.

The crux of our investigation is how to eliminate all covert channels [24] from communication channels in a mix network. For concreteness, imagine malware that leaks the permutation applied by a corrupted mix server by encoding this permutation in the publicly available random strings associated with mixing (whether in the ciphertexts or the proof of robustness.) Or, conversely, consider malware that applies to a set of inputs a permutation that looks random, but is known to and chosen by the attacker.

To prevent such attacks, we need to both ensure that no covert channels can be established[4], as well as ascertain that no "exterior randomness" is used in place of the intended "interior randomness". Technically, these two requirements translate to exactly the same issue: the ability to audit that all the randomness used by a server was correctly generated by the server's on-board pseudo-random generator. This must be done without exposing the actual state of the pseudo-random generator, since we do not wish to have to place any trust in observers – other than the assumption that at least one of them would truthfully alert the community if any irregularity is detected.

To protect the *observers* from attacks aimed at suppressing their activity, we use the notion of *undercover* observers (as introduced in [3]). Undercover observers are network participants that verify non-interactive proofs (or witnesses) of consistent generation and use of randomness, and which do not need to advertise their existence until they detect cheating and raise an alert.

The construction of witnesses of correct randomness is made difficult by the fact that these must not reveal what randomness is used, but must still eliminate covert channels with all but an exponentially small probability, and must not introduce covert channels themselves. In particular, this makes most of the recently developed techniques for efficient mixing unsuitable, since there is no apparent way to prove a disjunction in a way that (a) uses only pre-committed random strings, and (b) does not reveal what component of the proof the prover knows a witness to. (However, we will show that our proposed technique in fact can be used to implement such a proof, by ways of first implementing a mix network that has the property.)

Also, it is interesting to note that the traditional use of cut-and-choose techniques is not suitable either. It is clear that commitments *that are not opened* can trivially be made to leak a logarithmic amount of information (in the length of the commitment). In a situation with binary challenges, this allows an attacker to select one commitment in a $(2 \times k)$ matrix of commitments, and use the selected commitment to leak the information in question. Since this commitment will only be audited with a 50% probability, this corresponds to a success rate of an attacker of 50%. While it is easy to reduce this success rate, we note that a success rate that is polynomial in the length of the transcripts is not desirable. However, defying the intuition associated with this example, we show how to use vectors of homomorphic commitments to generate witnesses that defend against attacks with all but an exponential probability. This is applied both to re-encryption exponents and to permutations (as either could potentially be used to implement a covert channel.) More precisely, we introduce a method by which commitments are tied together in a pairwise manner, and where it is impossible to modify *either* of the two committed values without this being detected when *only one of them* is opened up.

While we base our design on a mix network construction that is not highly efficient [18], we note that the overhead caused by the addition of our security

[4] For practical reasons, we do not consider timing channels; we will discuss this later on.

measures is *minimal.* In spite of the difficulties to design protocols that implement tamper-evidence, we see no reason why more efficient designs could not be feasible. Thus, tamper-evidence is not a theoretical curiosity, but a practically achievable goal. It is our hope that tamper-evidence will become a mainstream design feature of any protocol that is potentially vulnerable to coercive attacks, particularly in the context of electronic elections.

Organization of the paper. We begin by outlining related work (section 2), followed by a description of our model and requirements (section 3), and a brief overview of re-encryption mix networks (section 4). In section 5, we present collapsed Merkle hash trees that will serve as a building block for our protocol. Finally, we present our tamper-evident mix network protocol in section 6, together with relevant proofs.

2 Related Work

The concept of covert channels in cryptographic protocols was introduced in 1983 in the seminal work of Simmons [24, 25]. Simmons specifically demonstrated the use of the Digital Signature Standard (DSS) signature scheme for covert communication. This showed that a secrete message could be hidden inside the authenticator. Young and Yung [29, 30] later showed the existence of covert channels in the key establishment algorithms of signature schemes.

Desmedt [4] presented a practical authentication scheme free of covert-channels, in which an observer (named "active warden") intercepts all the messages exchanged between two parties and verifies that they are free from covert information before passing them on. The observer defined by Desmedt is "active" in the sense that it interacts with the communicating parties. In contrast, [3] defines signature schemes which can be verified free of covert channels by *undercover* observers. Undercover observers verify signatures non-interactively, so that their stealthy existence can remain a secret at least up until the point when they detect an incorrect signature and raise an alarm. Undercover observers are preferable to active observers, because they are far less vulnerable to attacks aimed at suppressing their activity. This paper adopts the model of undercover observers of [3], but considers the far more complicated problem of ensuring the covert-free (or "tamper-evident") operation of a mix network.

To motivate our tamper-evident mixnet construction, we review briefly other mix networks in the literature and highlight the difficulties in making them tamper-evident. As a first example, consider the mix network recently proposed by Jakobsson, Juels and Rivest [10], and later used in the election scheme put forward by Chaum [2]. Therein, each mix server re-encrypts and permutes a list of n input ciphertexts *two times*, and commits to the ciphertext values in-between the two rounds. Then, a set of verifiers selects some n challenges[5] ; if the i^{th} challenge is a zero (resp. one) then the computation resulting in (resp.

[5] As usual, this step can be replaced by a random oracle if the Fiat Shamir heuristic is employed for challenge selection.

starting from) the i^{th} ciphertext value in-between the two rounds is revealed. Whereas this method does not result in the maximum anonymity set (namely n), it still provides reasonable anonymity for many applications, and at a very low cost. There is no straightforward way to design a tamper-evident variant of this scheme, since pairs of commitments (to the left and to the right) do not have the property that the modification of one of the values invalidates *both* commitments. This results in a success probability of 50% for an adversary. This can trivially be limited to $1/k$ if one were to employ k successive rounds of re-encryption and permutation. The cost of this, though, would be linear in k.

Turning now to a second (and rather common) class of mix network constructions, let us take a brief look at a scheme suggested by Abe [1]. Therein, the inputs are broken up in pairs, each one of which is mixed; the resulting list of ciphertexts are then (deterministically) shifted around and paired up, and the resulting pairs are mixed. This is repeated a logarithmic number of times in the number of input ciphertexts. In each mix-of-two, it is proven that either the identity permutation is used, or the “cross-over” permutation – along with corresponding proofs of correct re-encryption. This type of construction therefore employs disjunctive proofs. While we can construct tamper-evident disjunctive proofs using our proposed mix network scheme, we have not been able to find any simple (and inexpensive) construction for disjunctive proofs. Naturally, the same holds for disjunctive proofs involving larger number of inputs. Thus, this class of mix network schemes are not easily adopted to implement tamper-evidence.

Given that electronic elections is one of our motivating applications, it is meaningful to consider the impact of our approach in such settings. An interesting example of a situation in which our construction has an immediate impact is the coercive attack proposed by Michels and Horster [16] in which an attacker succeeds in verifying the value of a cast vote by corrupting both a voter and some random subset of mix servers. If an approach like ours is deployed (and the model changed correspondingly) then such an attack will be detected, and thus, will fail.

Some approaches, such as [17, 22], allow servers to verify each other’s actions to avoid leaks of secret information (such as random permutations.) Our approach, in contrast, prevents the *replacement* of the state of the pseudo-random generator. Moreover, and in comparison to these efforts, our scheme reduces the threats associated with potential covert channels caused by use of interaction.

The strongest relation to previous work is found in the collusion-free protocols defined by Lepinksi, Micali, and Shelat [14], which allow for the detection of collusion by malicious participants during the execution of the protocol. Our proposed scheme can be considered as the first practical implementation of collusion-free protocol for mix-networks. While [14] presents a well-defined abstract structure for collusion-free protocols, its application to mix networks is not obvious, in particular given the need to retain privacy. From that point of view, our contribution is to present a practical implementation eliminating collusions, i.e., possibilities to build covert-channels while maintaining privacy guarantees.

3 Model

Participants. We consider the following entities: *users*, *servers*, and *observers*. In addition, we assume the existence of an *authority* and an *attacker*.

- The users generate ciphertexts and post these to a public bulletin board $\mathcal{BB}$.
- Sequentially, the servers read the contents of $\mathcal{BB}$ and process portions of its contents in batches, writing the results of the operation back to $\mathcal{BB}$. These results consist of a set of ciphertexts (that constitute the inputs to the next server in the sequence) and a witness of tamper-freeness. The witness of tamper-freeness constitutes evidence that the (pseudo) random source of the server was not tampered with, and that the operation of the server correctly proceeded according to that random source. As we shall see, the witness of tamper-freeness implies the correctness of the mixing operation, and thus our servers do not need to provide an additional proof of correct mixing.
- The *attacker* is allowed to corrupt all but two of the users all of the time; this is corruption in the standard cryptographic sense, involving full read and write access to the compromised machines. The servers are also able to corrupt *all* of the servers all the time except during the key generation phase; this corruption allows full write access to compromised machines, but requires any information to be read from the machine to be transmitted using the standard communication media (as opposed to a secret side-channel). Thus, it is assumed that an attacker can send messages to corrupted servers out of band, but that all communication in the opposite direction (from a corrupted server to the attacker) must utilize the $\mathcal{BB}$, to which all servers have constant read and (appendive) write access. The latter is not a standard cryptographic assumption, but corresponds to realistic attacks in which software is corrupted by a "remote" attacker able to inject or replace code, e.g., by means of malware. Finally, the attacker is assumed able to corrupt (in the standard sense) all but one observer all of the time.
- The *observers* access $\mathcal{BB}$ and verify the correctness of witnesses posted thereon; if any witness is invalid (or missing) then any uncorrupted observer will initiate an alert.
- When an alert occurs, the *authority* will verify the validity of the alert (that it was done in accordance with the protocol specifications) and then physically disconnect any server whose witness was found to be invalid or missing[6].

Note that our techniques do not protect against timing covert-channels. However, by imposing strict requirements on synchronization or introducing random delays, one can protect against timing attacks as well, at the cost of a somewhat reduced (but predictable) throughput.

[6] We are mainly concerned with *detection*. After such detection, one can act on that information using standard methods, such as emulation or replacement of the faulty servers.

Goals.

- **Correctness/robustness.** The goal of the honest servers is to generate an output that consists of a set of ciphertexts, with a one-to-one correspondence to the batch of input ciphertexts given as input to the sequence of servers. Two ciphertexts must both decrypt to the same plaintext in order for us to say that they correspond to each other.
- **Privacy.** The goal of the attacker is to determine the mapping between input and output ciphertexts (for input ciphertexts not generated by users he has corrupted) with a probability of success that is significantly better than what could be achieved by a guess made uniformly at random from the possible mappings; or to extract information from a server that allows it to be impersonated with a probability of success that is significantly better that the probability of success that can be achieved without corruption of any servers[7].
- **Tamper-evidence.** The goal of the observers is to detect the use of any randomness inconsistent with the initial state of the corresponding server. This effectively corresponds to preventing covert communication and avoiding that the output of a corrupted server is a non-trivial function of information communicated to it by the attacker.

Trust. For the correctness property to hold, it is normally required that a majority of mix servers are honest. In our setting, though, it suffices that one observer and the authority are uncorrupted[8].

Similarly, for the privacy property to hold in our proposed scheme, no trust assumptions need to be made of either users or servers, but we have to assume that at least one observer and the authority are uncorrupted. If we recall that the main role of the observer is to detect inconsistent use of the randomness used for privacy, we can provide *correctness against privacy abuse* to build covert channel.

We do not need to trust any server with keeping any secret information of any other server. We assume that the authority will promptly disconnect any server failing to generate and output a valid witness for each transcript it writes to the bulletin board.

[7] We note that the second goal does not necessarily subsume the first. Consider, for example, a re-encryption mix network in which each server authenticates its output using its secret key. Knowledge of this key will not allow the attacker to determine the permutation, but knowledge of the state of pseudo-random generator does. In contrast, if we consider a decryption mix server based on padded RSA ciphertexts, it is clear that knowledge of the secret key will allow an adversary to infer the permutation.

[8] Alternatively, the correctness property can be seen to hold in a slightly different model in which there is no authority. Then, the requirement is instead that at least one observer is uncorrupted, and that all consumers of information pay attention to alerts.

Remark 1: Note that we make two simultaneous and different trust assumptions on servers. As far as tamper-evidence is concerned, we assume that the servers are honest (i.e., not corrupted) during the key generation phase. However, in terms of the protocol robustness, we do not make this assumption. This means that our protocol remains robust even if servers are corrupted during the key generation phase, whereas the same does not hold for tamper-evidence.

Remark 2: We note that in the following, we only address how to make re-encryption mixing tamper-evident. In most applications involving re-encryption mix networks, there is a phase involving decryption of output ciphertexts. We may assume that this functionality (which can be made tamper-evident following the techniques presented in [3]) can be blocked by the authority in the case of an alert. Practically speaking, this will be possible if a sufficient number of decryption servers can be disconnected immediately upon detection of an irregularity in the mix phase. In the following, we focus solely on the re-encryption mix process, and do not address the decryption process any further.

4 Preliminaries

We give a brief overview of re-encryption mix networks [18] based on the ElGamal cryptosystem (a more detailed description can be found, e.g., in [6]):

- **Key generation :** let p and q be primes such that $q \mid (p-1)$ and let $g \in \mathbb{Z}_p^*$ be an element of order q, such that the ElGamal cryptosystem defined by g in $\mathbb{Z}_p^*$ is semantically secure against plaintext attacks [32] and also adaptive chosen plaintext attacks. Consider a (t, l)-threshold encryption scheme [7] where the secret key is shared among l mix-servers. For $i = 1, \cdots, l$, mix-server $\mathcal{S}_i$ has secret key $x_i \in \mathbb{Z}_q^*$ and publicizes the corresponding public key $y_i = g^{x_i} \mod p$. Let $y = \prod_{i=1}^{l} y_i \mod p$.

- **Batch generation :** Let m_j denote the plaintext input of user U_j for $j = 1, \cdots, n$. The ElGamal encryption of m_j is

$$\mathsf{Enc}(m_j, r_j) \triangleq (g^{r_j}, y^{r_j} m_j),$$

  where $r_j \in \mathbb{Z}_q^*$ is chosen uniformly at random. Let the ciphertext be $(a_j, b_j) = \mathsf{Enc}(m_j, r_j)$. Each user U_j submits the ciphertext (a_j, b_j) as well as a proof of knowledge for the corresponding plaintext m_j (See [8]).

- **Mixing phase :** Each mix-server $\mathcal{S}_i$ performs two operations: re-encryption and permutation. More precisely, server $\mathcal{S}_i$ takes as input a list of n ciphertexts $((a_1, b_1), \cdots, (a_n, b_n))$ from $\mathcal{BB}$. For $j = 1, \ldots, n$, server $\mathcal{S}_i$ re-encrypts input (a_j, b_j) as follows:

$$(a'_j, b'_j) = \mathsf{ReEnc}\left((a_j, b_j), \alpha_j\right) \triangleq \left(g^{\alpha_j} \cdot a_j,\ y^{\alpha_j} \cdot b_j\right),$$

where α_j is a re-encryption parameter chosen at random in $\mathbb{Z}_q^*$. Server $\mathcal{S}_i$ then chooses a random permutation π on $\{1, 2, \cdots, l\}$ and outputs to $\mathcal{BB}$ the permuted list $\left(\left(a'_{\pi(1)}, b'_{\pi(1)}\right), \cdots, \left(a'_{\pi(n)}, b'_{\pi(n)}\right)\right)$.

– **Decryption phase :** A quorum of mix servers can do a threshold decryption of the final set of outputs, which yields the set of inputs $(m_1, \cdots, m_n)$ permuted according to the successive permutations applied by the l mix servers.

5 Building Block - Merkle Hash Tree Verification

A Merkle tree [15] is a tree consisting of nodes whose values are a one-way hash function (for example, SHA-1 or MD5) of the values of their children nodes. Due to their simplicity, Merkle trees are used for a wide range of secure authentication schemes. A Merkle tree is generally a binary tree where the value at a node N in the tree is defined with respect to the values N_{left} and N_{right} of its children by

$$N \triangleq h(N_{left} \,||\, N_{right})$$

where h denotes a one-way hash function and "$||$" denotes concatenation.

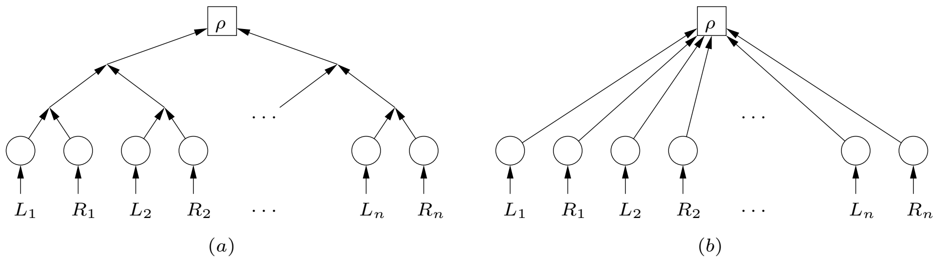


Fig. 1. A binary Merkle hash (a) and collapsed Merkle hash tree (b)

For better efficiency, our protocol does not use binary Merkle trees, but instead collapsed Merkle hash trees in which $2n$ leaves are connected to the root of the tree directly as shown in Fig. 1(b). In our protocol, the $2n$ leaves of the collapsed Merkle hash tree will be the elements of two sets $(L_1, \cdots, L_n)$ and $(R_1, \cdots, R_n)$ each of size n. We define a function $\mathsf{MerTree}$ that takes these sets as inputs and outputs the root ρ of the corresponding collapsed Merkle hash tree:

$$\mathsf{MerTree}((L_1, \cdots, L_n), (R_1, \cdots, R_n)) \triangleq h\Big(h(L_1) \,||\, h(R_1) \,||\, \cdots \,||\, h(L_n) \,||\, h(R_n)\Big)$$

The root ρ of the tree functions as a commitment to the sets $(L_1, \cdots, L_n)$ and $(R_1, \cdots, R_n)$. Note that this commitment can be verified given:

– either $(L_1, \cdots, L_n)$ and $(h(R_1), \cdots, h(R_n))$
– or $(h(L_1), \cdots, h(L_n))$ and $(R_1, \cdots, R_n)$.

6 Tamper-evident Mix Network

We propose a tamper-evident mix network in which each mix server pre-generates a random permutation together with a sequence of random re-encryption parameters that will be used to re-encrypt and mix the input batch. As explained in section 3, we assume that the generation of these parameters occurs in a setup phase prior to mixing, during which the mix servers are uncorrupted. During the mixing phase, the mix server outputs a proof that it operates in accordance with pre-generated parameters. Any deviation from these parameters invalidates the corresponding proof with all but negligible probability. This mix-network protocol thus ensures that the operation of mix-servers is tamper-evident.

Key generation. As explained in section 4, the l mix servers jointly generate the secret and public parameters for a (t, l)-threshold ElGamal encryption scheme. The public parameters are two primes p and q such that $q|(p-1)$ and an element $g \in \mathbb{Z}_p^*$ of order q. For $i = 1, \ldots, l$, we let $x_i \in \mathbb{Z}_q^*$ denote the secret key of mix-server $\mathcal{S}_i$ and $y_i = g^{x_i} \mod p$ the corresponding public key. We let $y = \prod_{i=1}^{l} y_i \mod p$.

Let κ be a security parameter, such that $2^{-\kappa}$ constitutes an acceptable error probability (for example $\kappa = 80$). To prove tamper-evident mixing, each server $\mathcal{S}_i$ generates additional values as follows. For notational clarity, we omit the suffix i, but it should be clear that each server generates its own set of the following values:

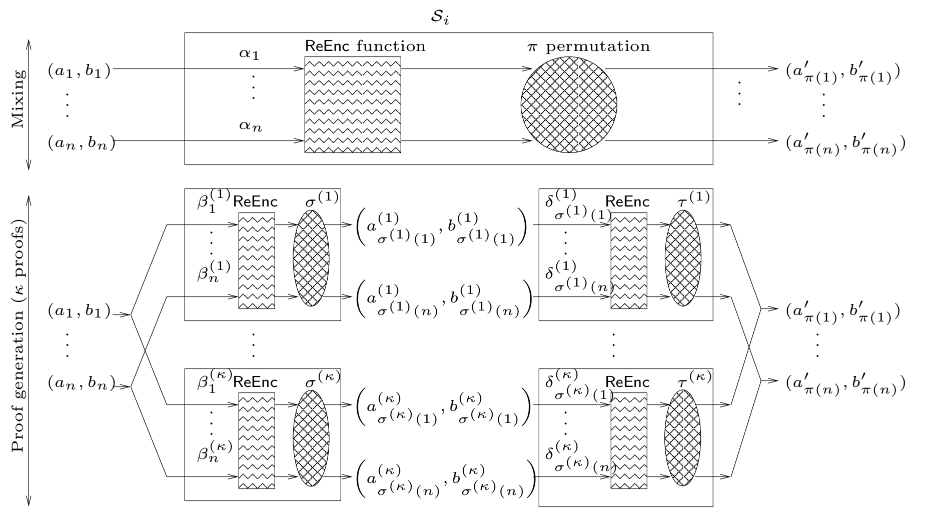
- a random permutation π on n elements
- n random values $\alpha_j \in \mathbb{Z}_q^*$ $(j = 1, \cdots, n)$ which are used as re-encryption parameters in the mixing phase
- κ pairs of permutations on n elements $\left(\sigma^{(1)}, \tau^{(1)}\right), \ldots, \left(\sigma^{(\kappa)}, \tau^{(\kappa)}\right)$ such that $\pi = \tau^{(k)} \circ \sigma^{(k)}$ for all $k = 1, \cdots, \kappa$. (As notational simplicity, we will continue to represent the index k in the superscripted braces.)
- κn pairs of integers $\left(\beta_j^{(k)}, \delta_j^{(k)}\right) \in \mathbb{Z}_q^* \times \mathbb{Z}_q^*$ such that $\alpha_j = \beta_j^{(k)} + \delta_j^{(k)}$ for all $j = 1, \ldots, n$ and $k = 1, \cdots, \kappa$.

The mix then computes commitments to the values $\sigma^{(k)}, \tau^{(k)}, \beta_j^{(k)}, \delta_j^{(k)}$ using collapsed Merkle hash trees. More precisely, the mix server constructs κ collapsed Merkle hash trees $T^{(1)}, \ldots, T^{(\kappa)}$. For $k = 1, \cdots, \kappa$, the leaves of $T^{(k)}$ consist of the following $2n + 2$ values in this order:

$$\sigma^{(k)}, \tau^{(k)}, \left(\beta_1^{(k)}, \ldots, \beta_n^{(k)}\right), \left(\delta_{\pi(1)}^{(k)}, \delta_{\pi(n)}^{(k)}\right).$$

We let $\rho^{(k)}$ denote the root of $T^{(k)}$. Each mix-server publicizes the root values $\rho^{(1)}, \ldots, \rho^{(\kappa)}$ of its Merkle trees.

Batch generation Each user U_j $(j = 1, \cdots, n)$ encrypts its plaintext message m_j by using group ElGamal encryption as described in Section 3 and posts the corresponding ciphertext to $\mathcal{BB}$.



$\mathcal{S}_i$
Mixing
ReEnc function
π permutation
α_1
α_n
(a_1, b_1)
(a_n, b_n)
$(a'_{\pi(1)}, b'_{\pi(1)})$
$(a'_{\pi(n)}, b'_{\pi(n)})$
Proof generation (κ proofs)
$\beta_1^{(1)}$ ReEnc $\sigma^{(1)}$
$\beta_n^{(1)}$
$\left(a^{(1)}_{\sigma^{(1)}(1)}, b^{(1)}_{\sigma^{(1)}(1)}\right)$
$\left(a^{(1)}_{\sigma^{(1)}(n)}, b^{(1)}_{\sigma^{(1)}(n)}\right)$
$\delta^{(1)}_{\sigma^{(1)}(1)}$ ReEnc $\tau^{(1)}$
$\delta^{(1)}_{\sigma^{(1)}(n)}$
$\beta_1^{(\kappa)}$ ReEnc $\sigma^{(\kappa)}$
$\beta_n^{(\kappa)}$
$\left(a^{(\kappa)}_{\sigma^{(\kappa)}(1)}, b^{(\kappa)}_{\sigma^{(\kappa)}(1)}\right)$
$\left(a^{(\kappa)}_{\sigma^{(\kappa)}(n)}, b^{(\kappa)}_{\sigma^{(\kappa)}(n)}\right)$
$\delta^{(\kappa)}_{\sigma^{(\kappa)}(1)}$ ReEnc $\tau^{(\kappa)}$
$\delta^{(\kappa)}_{\sigma^{(\kappa)}(n)}$

Fig. 2. Overview - A mix-server $\mathcal{S}_i$ re-encrypts the input (a_j, b_j) by $\mathsf{ReEnc}((a_j, b_j), \alpha_j)$ for $j = 1, \cdots, n$ and outputs $(a'_{\pi(j)}, b'_{\pi(j)})$ which is permuted according to the permutation π. In the meanwhile, the mix-server computes as proof κ sets of values; for $k = 1, \ldots, \kappa$, the prover $\mathcal{P}$ computes (for $j = 1, \ldots, n$) the values $\left(a_j^{(k)}, b_j^{(k)}\right) = \mathsf{ReEnc}\left((a_j, b_j),\ \beta_j^{(k)}\right)$ and outputs $\mathcal{W}^{(k)} = \left(a^{(k)}_{\sigma^{(k)}(1)}, b^{(k)}_{\sigma^{(k)}(1)}\right), \ldots, \left(a^{(k)}_{\sigma^{(k)}(n)}, b^{(k)}_{\sigma^{(k)}(n)}\right)$.

Mixing phase For $i = 1, \cdots, l$, mix-server S_i reads from $\mathcal{BB}$ the list of n ciphertexts output by S_{i-1} (the first server $\mathcal{S}_1$ gets from $\mathcal{BB}$ the list of n inputs ciphertexts submitted by the n users). We denote this list of ciphertexts by $((a_1, b_1), \ldots, (a_n, b_n))$. For $j = 1, \ldots, n$, server $\mathcal{S}_i$ re-encrypts the input (a_j, b_j) as follows:

$$\left(a'_j, b'_j\right) = \mathsf{ReEnc}\left((a_j, b_j), \alpha_j\right),$$

where the α_j are the values generated by $\mathcal{S}_i$ in the key generation phase (again, we omit the index i for notational clarity). The mix server $\mathcal{S}_i$ then outputs these values permuted according to the permutation π, i.e.

$$\left(a'_{\pi(1)}, b'_{\pi(1)}\right), \cdots, \left(a'_{\pi(n)}, b'_{\pi(n)}\right).$$

Proof of tamper-evidence. The mixnet outputs a witness of tamper-freeness. This witness is computed non-interactively. However, the construction of the witness is easier to understand if we describe it in terms of an interaction between a prover $\mathcal{P}$ (the mix-server) and a verifier $\mathcal{V}$ (the observers). It will be immediately clear that this interactive protocol can be turned into a non-interactive witness using the Fiat-Shamir heuristic.

1. **(Commitments)** For $k = 1, \ldots, \kappa$, the prover $\mathcal{P}$ computes (for $j = 1, \ldots, n$) the values
$$\left(a_j^{(k)}, b_j^{(k)} \right) = \mathsf{ReEnc}\left((a_j, b_j),\ \beta_j^{(k)} \right) \tag{1}$$
and outputs
$$\mathcal{W}^{(k)} = \left(a^{(k)}_{\sigma^{(k)}(1)}, b^{(k)}_{\sigma^{(k)}(1)} \right), \ldots, \left(a^{(k)}_{\sigma^{(k)}(n)}, b^{(k)}_{\sigma^{(k)}(n)} \right) \tag{2}$$

2. **(Challenges)** The verifier $\mathcal{V}$ outputs κ random challenges $c^{(1)}, \ldots, c^{(\kappa)} \in \{0, 1\}$.

3. **(Response to challenges)** For $k = 1, \cdots, \kappa$:
   - If $c^{(k)} = 0$: $\mathcal{P}$ outputs $\sigma^{(k)}$, $h\left(\tau^{(k)}\right)$, $\beta_1^{(k)}, \ldots, \beta_n^{(k)}$ and $h\left(\delta^{(k)}_{\pi^{(k)}(1)}\right), \ldots, h\left(\delta^{(k)}_{\pi^{(k)}(n)}\right)$
   - If $c^{(k)} = 1$: $\mathcal{P}$ outputs $h\left(\sigma^{(k)}\right)$, $\tau^{(k)}$, $h\left(\beta_1^{(k)}\right), \ldots, h\left(\beta_n^{(k)}\right)$ and $\delta^{(k)}_{\pi^{(k)}(1)}, \ldots, \delta^{(k)}_{\pi^{(k)}(n)}$

4. **(Verification)** For $k = 1, \cdots, \kappa$, the verifier checks the following depending on the value of $c^{(k)}$:
   - If $c^{(k)} = 0$: $\mathcal{V}$ re-encrypts input (a_j, b_j) with re-encryption factors $\beta_j^{(k)}$, then permutes them according to permutation $\sigma^{(k)}$ and checks that the result matches the set $\mathcal{W}^{(k)}$ received from $\mathcal{P}$ in Step 1.
   - If $c^{(k)} = 1$: $\mathcal{V}$ re-encrypts output $\left(a'_{\pi(j)}, b'_{\pi(j)} \right)$ with re-encryption factors $-\delta^{(k)}_{\pi^{(k)}(j)}$, then permutes them according to the inverse of permutation $\tau^{(k)}$ and checks that the result matches the set $\mathcal{W}^{(k)}$ received from $\mathcal{P}$ in Step 1.

   Finally, $\mathcal{V}$ reconstructs the collapsed Merkle hash tree $T^{(k)}$ and verifies that the root of that tree is equal to the root $\rho^{(k)}$ output by server $\mathcal{S}_i$ in the key generation step. It should be clear that the values output at the end of step 3 enable $\mathcal{V}$ to reconstruct the Merkle hash tree $T^{(k)}$ regardless of whether $c^{(k)} = 0$ or $c^{(k)} = 1$.
   If any of the verification steps fails, the verifier $\mathcal{V}$ raises an alarm and the prover (i.e. mix-server $\mathcal{S}_i$) is discarded.

Non-interactive proof of tamper-evidence. The interactive protocol given above to verify the tamper-freeness of a mix server's operation can be transformed into a non-interactive protocol with the Fiat-Shamir heuristic (also known as the random oracle model): the κ challenges in Step 1 of the proof can be replaced by the κ left-most bits of the hash of $\left((a'_{\pi(1)}, b'_{\pi(1)}), \cdots, (a'_{\pi(n)}, b'_{\pi(n)}) \right)$. A non-interactive protocol allows proofs of tamper-freeness to be verified by *undercover* observers. Undercover observers need not reveal their existence until they detect an incorrect proof and raise an alarm.

Decryption phase The final outputs of the mix server is decrypted by a quorum of mix servers. Quorum ElGamal decryption can be made tamper-evident following the techniques of [3].

6.1 Properties

Proposition 1. *If the hash function h is second pre-image resistant, then a dishonest prover $\mathcal{P}$ can cheat the verifier $\mathcal{V}$ with probability at most $2^{-\kappa}$ in the proof of tamper-evidence.*

Proof. The proof is by contraposition. Let us assume the existence of a prover $\mathcal{P}$ who can cheat the verifier $\mathcal{V}$ in the proof of tamper-evidence with probability $2^{-\kappa} + \epsilon$. We show how to use $\mathcal{P}$ to find a second pre-image for the function h. We proceed as follows:

1. $\mathcal{V}$ executes one instance of the key generation step described in Section 6. In particular, $\mathcal{V}$ outputs the roots $\rho^{(1)}, \ldots, \rho^{(\kappa)}$ of κ collapsed Merkle hash trees $T^{(1)}, \ldots, T^{(\kappa)}$. $\mathcal{V}$ also outputs a batch of n input ciphertexts $((a_1, b_1), \ldots, (a_n, b_n))$.
2. $\mathcal{P}$ outputs a new batch of n ciphertexts $((a''_1, b''_1), \ldots, (a''_n, b''_n))$, such that there exists at least one index $j \in \{1, \ldots, n\}$ such that
$$(a''_j, b''_j) \neq \mathsf{ReEnc}\left((a_{\pi(j)}, b_{\pi(j)}), \alpha_{\pi(j)}\right).$$
This condition expresses the assumption that $\mathcal{P}$ is a dishonest prover.
3. $\mathcal{P}$ outputs commitments $\mathcal{W}^{(k)}$ for $k = 1, \ldots, \kappa$. We distinguish two cases:
   - If $\mathcal{W}^{(k)}$ is *not equal* to the re-encryption of the inputs (a_j, b_j) with re-encryption factors $\beta_j^{(k)}$ permuted according to permutation $\sigma^{(k)}$, then we say that the commitment $\mathcal{W}^{(k)}$ is "input-incorrect" and we define $\overline{c}^{(k)} = 1$.
   - Otherwise, the commitment $\mathcal{W}^{(k)}$ is *equal* to the re-encryption of the inputs (a_j, b_j) with re-encryption factors $\beta_j^{(k)}$ permuted according to permutation $\sigma^{(k)}$. But since $(a''_j, b''_j) \neq \mathsf{ReEnc}\left((a_{\pi(j)}, b_{\pi(j)}), \alpha_{\pi(j)}\right)$, it must then be the case that $\mathcal{W}^{(k)}$ is *not equal* to the re-encryption of the outputs $(a'_{\pi(j)}, b'_{\pi(j)})$ with re-encryption factors $-\delta^{(k)}_{\pi^{(k)}(j)}$ permuted according to the inverse of permutation $\tau^{(k)}$. We say then that the commitment $\mathcal{W}^{(k)}$ is "output-incorrect" and we define $\overline{c}^{(k)} = 0$.
4. The verifier $\mathcal{V}$ outputs κ random challenges $c^{(1)}, \ldots, c^{(\kappa)} \in \{0, 1\}$.
5. The prover responds to these challenges and the responses are verified by $\mathcal{V}$ as described in the protocol of section 6.

With probability $2^{-\kappa}$, we have $\overline{c}^{(k)} = c^{(k)}$ for all $k = 1, \ldots, \kappa$. The prover, however, succeeds in convincing the verifier with probability $2^{-\kappa} + \epsilon$. It follows that with probability ϵ, the prover succeeds in convincing the verifier when there exists an index k such that $\overline{c}^{(k)} \neq c^{(k)}$. In this case, we show how to compute a second pre-image for the function h.

Without loss of generality, let us assume that for some $k \in \{1, \ldots, \kappa\}$, we have $\overline{c}^{(k)} = 1$ and $c^{(k)} = 0$. In other words, the commitment $\mathcal{W}^{(k)}$ is "input-incorrect", and $\mathcal{V}$ verifies the relationship between the inputs and the commitment. The root $\rho^{(k)}$ of the Merkle tree $T^{(k)}$ commits $\mathcal{P}$ to the values $\sigma^{(k)}, \tau^{(k)}$, $\beta_1^{(k)}, \ldots, \beta_n^{(k)}$ and $\delta_1^{(k)}, \ldots, \delta_n^{(k)}$. Let us denote $\sigma^{'(k)}, \tau^{'(k)}$, $\beta_1^{'(k)}, \ldots, \beta_n^{'(k)}$ and $\delta_1^{'(k)}, \ldots, \delta_n^{'(k)}$ the values used by $\mathcal{P}$ to compute $\mathcal{W}^{(k)}$. Let us denote $T^{'(k)}$ the collapsed Merkle tree computed with these alternate values and let $\rho^{'(k)}$ be the root of that tree. We know two things:

- The commitment $\mathcal{W}^{(k)}$ is "input-incorrect". Thus $\left(\sigma^{'(k)}, \beta_1^{'(k)}, \ldots, \beta_n^{'(k)}\right) \neq \left(\sigma^{(k)}, \beta_1^{(k)}, \ldots, \beta_n^{(k)}\right)$.
- The proof succeeds. Thus $\rho^{(k)} = \rho^{'(k)}$.

Thus we have used $\mathcal{P}$ to compute a second pre-image for the function h. □

Proposition 2. *Our mix network protocol is tamper-evident.*

Proof. This is an immediate corollary of Proposition 1. The proof of tamper-evidence ensures that the operation of every mix server is entirely deterministic based on the inputs committed to in the key generation step. □

Proposition 3. *Our mix network protocol guarantees correctness.*

Proof. The correctness of the mixing follows immediately from tamper-freeness. Indeed, we assume that correct re-encryption factors and permutations are selected in the key-generation phase, since the mix server is assumed uncorrupted during that phase (see our model and its justification in Section 3). □

7 Conclusion

Motivated by electronic elections, much research has been devoted to building mix networks that are secure against privacy threats, and whose operation can be verified correct. This paper introduces a new notion of security, which we call *tamper-evidence.* A mix server is tamper-evident if any variation from a prescribed deterministic mode of operation is detectable. The tamper-evident mix network scheme we propose extends the security requirements of mix networks to the run-time detection of covert channels (which constitute one kind of disallowed variation from the prescribed deterministic operation). The tamper-evidence of mix servers is verified by non-interactive observers, whom we call *undercover* observers. Undercover observers can operate stealthily (at least up to the point when they must raise an alarm) and are thus nearly impossible to detect and attack.

References

1. M. Abe. Mix-networks on permutation networks, In *ASIACRYPT '99*, LNCS 1716, Springer-Verlag, 1999. pp. 258–273.
2. D. Chaum. Secret Ballot Receipts: True Voter-Verifiable Elections. RSA CryptoBytes, Volume 7, No. 2, 2004.
3. J. Choi, P. Golle, and M. Jakobsson. Tamper-Evident Digital Signatures: Protecting Certification Authorities Against Malware. IACR ePrint report, No. 147, 2005.
4. Y. Desmedt. Subliminal-free authentication and signature. In *Advances in Cryptology – Eurocrypt '88*, LNCS 330. Springer-Verlag, 1988. pp. 23–33.
5. A. Fiat and A. Shamir. How to prove yourself: Practical Solution to Identification and Signature Problems. In *Advances in Cryptology – Crypto'86*, LNCS 26. Springer-Verlag, 1987. pp. 186–194.
6. P. Golle and M. Jakobsson. Reusable Anonymous Return Channels. In *Proc. of the Workshop on Privacy in the Electronic Society(WPES) '03*, ACM Press, 2003, pp. 94–100.
7. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Proc. of Eurocrypt '99*, LNCS 1592, Springer-Verlag, 1999. pp. 295–310.
8. M. Jakobsson. A practical mix. In *Proc. of Eurocrypt '98*, LNCS 1403, Springer-Verlag, 1998. pp. 448–461.
9. A. Juels and J. Guajardo. RSA Key Generation with Verifiable Randomness. In *Public Key Cryptography 2002*, LNCS 2274, Springer-Verlag, 2002. pp. 357–374.
10. M. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proc. of USENIX'02*, pp. 339–353.
11. M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo. Fractal Merkle Tree Representation and Traversal. In *Proc. of RSA Cryptographers' Track 2003*, 2003.
12. M. Jakobsson and M. Yung. Distributed "Magic Ink" Signatures. In *Advances in Cryptology – Eurocrypt '97*, LNCS 1233, Springer-Verlag, 1997. pp. 450–464.
13. C. Karlof, N. Sastry, and D. Wagner, Cryptographic Voting Protocols: A Systems Perspective. In *USENIX Security '05*, August 2005. pp. 33–50.
14. M. Lepinksi, S. Micali, and A. Shelat. Collusion-Free Protocols. In *STOC '05*, ACM Press, 2005.
15. R. Merkle. Secrecy, authentication, and public key systems. Ph.D. dissertation, Dept. of Electrical Engineering, Stanford Univ., 1979.
16. M. Michels and P. Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. In K. Kim and T. Matsumoto, editors, *ASIACRYPT '96*, LNCS 1163, Springer-Verlag, 1996.
17. C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proc. of CCS '01*. ACM Press, 2001. pp. 116–125
18. W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *Proc. of ICICS '97*, LNCS 1334, Springer-Verlag, 1997. pp. 440–444.
19. C. Park, K. Itho, and K. Kurosawa. All/Nothing Election Scheme and Anonymous Channel. In *Proceeding of Eurocrypt '93*, 1993.
20. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology – Eurocrypt '96*, LNCS 1070. Springer-Verlag, 1996. pp. 387–398.
21. Pedro A.D. Rezende. Electronic Voting Systems – Is Brazil Ahead of its Time?. RSA CryptoBytes, Volume 7, No. 2, 2004.
22. M. K. Reiter and X. Wang. Fragile Mixing. In *Proc. of CCS '04*, 2004. pp. 227–235
23. C. P. Schnorr. Efficient Signature Generation for Smart Cards. In *Proc. of Crypto '89*, 1989. pp. 239–252.
24. G. J. Simmons. The prisoners' problem and the subliminal channel. In *Proc. of Crypto '83*, 1983. pp. 51–67.
25. G. J. Simmons. The subliminal channel and digital signature. In *Proc. of Eurocrypt '84*, LNCS 209, Springer-Verlag, 1996. pp. 364–378.
26. M. E. Smid and D. K. Branstad. Response to comments on the NIST proposed Digital Signature Standard. In *Proc. of Crypto '92*, LNCS 740. Springer-Verlag, 1992. pp. 76–87.
27. M. Stadler, Publicly Verifiable Secret Sharing, In *Advances in Cryptology – Eurocrypt '96*, LNCS 1070, Springer-Verlag, 1996. pp. 190–199.
28. A. Young and M. Yung. The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone? In *Proc. of Crypto 1996*, 1996. pp. 89–103.
29. A. Young and M. Yung. The prevalence of Kleptographic attacks on discrete-log based cryptosystems. In *Proc. of Crypto '97*, 1997. pp. 264–276.
30. A. Young and M. Yung. Kleptography: using cryptography against cryptography. In *Proc. of Eurocrypt '97*, LNCS 1233, Springer-Verlag, 1997. pp. 62–74.
31. A. Young and M. Yung. Auto-Recoverable and Auto-Certifiable Cryptosystems. In *Advances in Cryptology – Eurocrypt '98*, LNCS 1403, Springer-Verlag, 1998. pp. 119–133.
32. Y. Tsiounis and M. Yung. On the Security of ElGamal Based Encryption. In *Proc. of PKC '98*, LNCS 1431, Springer-Verlag, Feb. 1998. pp. 117–134