Optimistic Mixing for Exit-Polls

Philippe Golle[1], Sheng Zhong[2], Dan Boneh[1],
Markus Jakobsson[3], and Ari Juels[3]

[1] Stanford University, Stanford, CA 94305 USA
{pgolle,dabo}@cs.stanford.edu
[2] Yale University, New Haven, CT 06520 USA
sheng.zhong@yale.edu
[3] RSA Labs, RSA Security Inc, Bedford, MA 01730 USA
{mjakobsson,ajuels}@rsasecurity.com

Abstract. We propose a new mix network that is optimized to produce a correct output very fast when all mix servers execute the mixing protocol correctly (the usual case). Our mix network only produces an output if no server cheats. However, in the rare case when one or several mix servers cheat, we convert the inputs to a format that allows "back-up" mixing. This back-up mixing can be implemented using any one of a wide array of already proposed (but slower) mix networks. When all goes well, our mix net is the fastest, both in real terms and asymptotically, of all those that offer standard guarantees of privacy and correctness. In practice, this benefit far outweighs the drawback of a comparatively complex procedure to recover from cheating. Our new mix is ideally suited to compute almost instantly the output of electronic elections, whence the name "exit-poll" mixing.

1 Introduction

The recently devised mix network constructions of Furukawa and Sako [FS01] and Neff [Nef01] provide the full spectrum of security properties desirable in an election scheme. They achieve privacy, which is to say concealment of individual votes, and also robustness against Byzantine server failures. They additionally possess the property of *universal verifiability*, that is, the ability for any entity to verify the correct functioning of the mix, even in the face of an adversary that controls all servers and voters. Finally, the Furukawa/Sako and Neff mixes are substantially more efficient in terms of both computational and communications requirements than previously proposed mix networks with similar security properties.

Fast as they are, however, these mixes still remain cumbersome as tools for large-scale elections. Sako *et al.* report a running time of roughly six hours to process a batch of 100,000 votes [FMMOS02]. In a federal election involving large precincts (conceivably millions of ballots in some states) a complete tally would thus require many hours. Premature media predictions of Gore's victory in Florida in the 2000 U.S. presidential election demonstrate the hunger of the electorate for timely information, and also the mischief that can be wrought in its absence. There is clearly a political and social need for faster tallying mechanisms than Furukawa/Sako and Neff alone can provide.

We describe here a mix network that is tailored for election systems, but with a substantial speedup over Furukawa/Sako and Neff. In settings like that described by Sako *et al.*, for example, we estimate that our construction is capable of yielding a six-to-eight times speedup. We achieve this improvement by taking an "optimistic" or "fast-track" [GRR98] approach. In particular, we identify functionality in Furukawa/Sako and Neff that is not needed in the likely case that mix servers behave correctly and that most ballots are well formed. In the optimistic case, we show how to dispense with the costly property of robustness against Byzantine server failures. We also provide a form of universal verifiability that is somewhat weaker than the standard definition, but less costly, and adequate for nearly all types of elections, as we explain.

We refer to our proposal as an *exit-poll* mix network, by analogy with the "exit polls" used to provide fast predictions of election outcomes. If servers behave correctly, our exit-poll mix yields a correct election tally very rapidly. We expect this to be by far the most common case. If server cheating occurs, our mix identifies misbehaving servers. The privacy of all votes remains protected (given a majority of honest servers), but our mix does not produce an output. In such cases, our exit-poll scheme permits seamless fallback to a more heavyweight mix (like Furukawa/Sako or Neff) which can take over, complete the mixing and produce an output. Such heavyweight mixes can also be employed to achieve supplemental, after-the-fact certification of an election tally achieved with our mix.

Our exit-poll mix is a general ciphertext-to-plaintext scheme. While it is designed particularly for use in election schemes, we note that it can be employed in many of the other applications for which such mix networks are useful. Examples include anonymous e-mail [Cha80] and bulletin boards, anonymous payment systems [JM98] as well as anonymized Web browsing [GGMM97].

The rest of the paper is organized as follows. We review related work in section 2. In section 3, we describe ElGamal re-encryption mix networks. We present the high-level design of our new mix network in section 4, and give a detailed description of the protocol in section 5. In section 6, we prove the properties of our mix net. We conclude in section 7.

2 Related Work

Chaum proposed the first mix network, a decryption mix, in [Cha80]. In Chaum's construction, users encrypt their inputs with the public-key of each mix server, starting with the last and ending with the first mix server in the net. Each mix server removes one layer of encryption, until the plaintexts are output by the last server. The weakness of this approach is that the mixing can not proceed if a single server is unavailable. To achieve robustness against server failures, [PIK93] introduced a new type of mix, re-encryption mix nets, in which the mixing and decryption phases are separated (see section 3). The particular re-encryption mix of [PIK93] was shown insecure in [PP89,Pfi94], but was later fixed by [OKST97].

The main difficulty of re-encryption mixes is to design computationally efficient ways for mix servers to *prove* that they mixed and re-encrypted their inputs correctly in the mixing phase. The first techniques were based on costly general purpose cut-and-choose zero-knowledge proofs [SK95,OKST97,Abe98]. Millimix [JJ99] and MIP-2 [Abe99,AH01] are based on more efficient zero-knowledge proofs specifically designed to prove that an output is a re-encryption of one of two inputs.

The most efficient schemes to date that offer the full spectrum of security properties are those of Furukawa and Sako [FS01] and Neff [Nef01]. The table in figure 1 compares the real cost of mixing n items (in terms of the number of exponentiations required) with different mixing schemes (the numbers are taken from the respective papers). The column indicating the cost of proof and verification is in bold, because that is typically by far the most expensive step, and it is the step that we are optimizing. The cost of re-encryption is higher in our scheme than in others, but the difference pales in comparison with our savings in the proof and verification step. Furthermore, the re-encryption exponentiations can be pre-computed. The table also indicates whether each mixing scheme can take advantage of the speed-up techniques proposed in [Jak99][4] for multiple exponentiations with respect to a fixed base. These techniques, based on addition chains, reduce the equivalent cost of one exponentiation to approximately 10 multiplications for reasonable sizes of batches (see [Jak99] for more details). This amounts to a very significant speed-up. Our scheme is not only the fastest, but also the only one that can fully take advantage of addition chains in this sense.

Scheme	Re-encrypt	**Proof and verification**	Decrypt	Addition chain speed-up?
Cut and choose[5][SK95,OKST97]	$2n$	$642nk$	$(2+4k)n$	no
Pairwise permutation[Abe99,JJ99]	$2n$	$7n \log n(2k-1)$	$(2+4k)n$	partially
Matrix representation[FS01]	$2n$	$18n(2k-1)$	$(2+4k)n$	partially
Polynomial scheme[Nef01]	$2n$	$8n(2k-1)$	$(2+4k)n$	partially
Exit-poll mixing [this paper]	$6n$	$6+12k$	$(5+10k)n$	yes

Fig. 1. Optimistic cost per server (for a total of k servers) of mixing n items with different mix schemes, measured in number of exponentiations. Note that our proof and verification costs do not depend on n. The table also indicates whether addition chains can be used to pre-compute exponentiations. "Partially" indicates that addition chains can be used only in the mixing phase but not to prove correctness.

[4] Other aspects of that proposal were later found flawed, and corrected, in [MK00]. The exposed vulnerabilities do not affect the soundness of the speed-up techniques.

An attractive alternative to mix networks is homomorphic encryption, in particular the Paillier scheme [Pai99]. Election schemes based on homomorphic encryption require a good deal of computation for verification of correct ballot formation, but very little for tallying. In practice, therefore, they can be much faster than mix-based election schemes. Until recently, an objection to homomorphic schemes has been their inability to accommodate write-in votes, an unavoidable requirement in the election systems of many jurisdictions. Kiayias and Yung have recently devised a simple scheme that circumvents this difficulty [KY02]. In brief, the idea is to permit each ballot to contain either a standard vote or a write-in vote, and to set aside write-in votes for separate processing via a mix network (in the unlikely case that this is needed).

It is our belief that mix networks will nonetheless remain an essential tool in electronic voting, as they still provide features that homomorphic schemes cannot. Vote-buying and coercion are serious threats in any election, but potentially much more problematic in Internet-based elections, given the anonymizing mechanisms available on the Internet and its reach across many jurisdictions. Schemes based on mix networks offer ways of minimizing these threats [HS00], while homomorphic schemes do not. A second advantage of mix networks is their flexibility with regard to key distribution. To distribute shares in the Paillier system without use of a trusted third party requires expensive joint RSA key-generation protocols (e.g., [BF97]), and distribution of a fresh RSA modulus for every election involving a different distribution of trust. Mix-based schemes can be based on discrete-log cryptosystems, with simpler and more generalizable keying mechanisms. With this in mind, we propose a new mix network which offers a significant efficiency improvement over existing constructions.

3 ElGamal Re-encryption Mix Network

In this section, we describe the basic operation of a plain-vanilla re-encryption mix network based on the ElGamal cryptosystem. It will serve as a basis for our main construction described in section 4 and 5. The operation of a mix network can be divided into the following steps:

1. **Setup phase.** In the setup phase, the mix servers jointly generate the public and private parameters of an ElGamal cryptosystem. The private key is shared in a (t, n)-threshold verifiable secret sharing scheme among all mix servers, while the public parameters are published.
2. **Submission of inputs.** All users submit their inputs to the mix encrypted with the public parameters generated in the setup phase.
3. **Mixing phase.** Each mix server in turn mixes and re-randomizes the batch of ciphertexts submitted to the mix.

[5] We note that these proposals have computational costs quadratic in the number of servers, due to the use of interactive proofs. However, if non-interactive proofs are employed – as in subsequent papers – this is brought down to a linear cost. The computational cost we use in the table assumes that this enhancement is performed.

4. **Decryption phase.** After the mixing is done, all output ciphertexts are decrypted by a quorum of mix servers.

We start with a description of the ElGamal cryptosystem, and discuss in particular how to re-randomize ciphertexts in the mixing phase. We then explain how to jointly generate the parameters for an ElGamal cryptosystem in the setup phase, and how the quorum decryption works in the decryption phase.

3.1 ElGamal Cryptosystem

ElGamal is a randomized public-key encryption scheme. Let P and Q be two large primes such that $P = 2Q + 1$. We denote by G_Q the subgroup of $\mathbb{Z}_P^*$ of order Q. Let g be a generator of G_Q. The private key is an element $x \in \mathbb{Z}_Q$, and the corresponding public key is $y = g^x \mod P$. To encrypt a plaintext $m \in G_Q$, we choose a random element $r \in \mathbb{Z}_Q$ and compute the ciphertext $E_y(m, r) = (g^r, my^r)$. Note that an ElGamal ciphertext is a pair of elements of G_Q. To get the decryption $D_x(G, M)$ of an ElGamal ciphertext (G, M), we compute $D_x(G, M) \overset{\triangle}{\equiv} M/G^x$. The ElGamal cryptosystem is semantically secure [Bon98] if the decisional Diffie-Hellman assumption holds in the group G_Q.

Re-randomization.
ElGamal is a randomized encryption scheme that allows for re-randomization of ciphertexts. Given an ElGamal ciphertext (G, M), a mix server can efficiently compute a new ciphertext (G', M') that decrypts to the same plaintext as (G, M) (we say that the ciphertext (G', M') is a re-randomization of (G, M)). To re-randomize a ciphertext, the mix server chooses a value $r \in \mathbb{Z}_Q$ uniformly at random and computes $(G', M') = (Gg^r, My^r)$. Observe that this does not require knowledge of the private key, and that the exponentiation can be pre-processed.

Given two ElGamal ciphertexts, it is infeasible to determine whether one is a re-randomization of the other without knowledge of either the private decryption key x or the re-randomization factor r, assuming that the Decision Diffie-Hellman problem is hard in G_Q. A mix server can use this property to hide the correspondence between its input and output ciphertexts: the input ciphertexts are first re-randomized, then output in a random order.

However, a mix server who knows the re-randomization factor r can efficiently convince a verifier that (G', M') is a re-randomization of (G, M) without revealing r. The proof of re-randomization consists simply of proving that $\log_g(G'/G) \equiv \log_y(M'/M) \pmod P$, which trivially implies that there exists r such that $(G', M') = (Gg^r, My^r)$. To prove the former discrete logarithm equality, we may use for example Schnorr signatures [Sch91] (as suggested in [Jak98]) or a non-interactive version [FS86] of the Chaum-Pedersen protocol [CP92]. This proof of re-randomization will serve as the basis for a proof that allows a mix server to prove that it mixed its inputs correctly (observe that in the real proof of correctness, a mix server must not reveal which output is a re-randomization of which input, so the proof outlined above will not work *as is*.)

3.2 Distributed ElGamal

In the setup phase, the mix servers jointly generate the parameters (P, Q, g, x, y) of an ElGamal cryptosystem, in such a way that the private key x is distributed in a (n, t)-threshold verifiable secret sharing (VSS) scheme among all mix servers [Fel87]. To set up this VSS, a simple solution is to have a trusted "dealer" generate all the parameters and then distribute shares of the private key to the mix servers. (An alternative solution that does not require a trusted third party was proposed in [Ped91], but it was later found flawed by [GJK+99]. Note that the proved-correct protocol suggested in [GJK+99] is for a different VSS scheme.)

With the private key thus shared, it is known that any quorum of t mix servers can jointly decrypt the output ElGamal ciphertexts without explicitly reconstructing the private key x. A quorum T of t servers can decrypt a ciphertext (G, M) as follows:

$$D_x(G, M) = \frac{M}{G^x} = \frac{M}{\prod_{j \in T} (G^{x_j})^{\prod_{l \in T, l \neq j} \frac{-l}{j-l}}}.$$

Observe that this equation requires each server $j \in T$ to raise G to the x_j-th power. Server j may prove that it has honestly computed $S = G^{x_j}$ with the following proof of discrete logarithm equality: $\log_G S \equiv \log_g y_j (= x_j) \pmod{P}$.

4 Mix Net Design

Our new mix net mixes ciphertexts like an ElGamal re-encryption mix. The novelty lies first in a highly efficient method for proving that the mixing was done correctly, and second in a method for falling back on a more heavyweight mix if cheating by a server is detected. We start with a high-level description of these two building blocks.

Each input ciphertext submitted to our mix net is required to be the encryption of a plaintext that includes a cryptographic checksum. To verify that a mix server operated correctly, we ask for a proof that the product of the plaintexts corresponding to the input ciphertexts equals the product of the plaintexts corresponding to the output ciphertexts. As we shall show, such proofs can be produced and verified highly efficiently without knowledge of the plaintexts. We call this proof a *proof-of-product (POP) with checksum.*

This proof however does not detect all types of cheating. Rather, it guarantees that if the mix server did not mix correctly, it had to introduce in the output at least one new ciphertext that corresponds to a plaintext with an invalid checksum. When outputs are decrypted, invalid checksums are traced to one of two sources: either an input that was originally submitted to the mix network with an invalid checksum, or a cheating mix server. The difficulty of this approach lies in the fact that since invalid checksums can only be traced at decryption time, cheating may not be detected until after the harm is done. In effect, a cheating server may be able to match inputs to outputs before cheating gets detected in the verification step. If we were to use this mix just like that, nothing could be done after a server has cheated to restore the privacy of those users whose inputs have already been traced. In particular, a second round of mixing wouldn't help.

To address this difficulty, we introduce the second main contribution of this paper, which may be of interest on its own. Our approach is to encrypt users' inputs twice (a technique we call *double enveloping*). In the verification step outlined above, the output ciphertexts are first decrypted only once. If the verification succeeds and no servers are found to have cheated, the output ciphertexts are decrypted one more time and yield the plaintext. If on the other hand one or several servers are found to have cheated, the output ciphertexts are not decrypted further. Instead, they become the input to a different (slower) mix network such as [Nef01] and are mixed a second time before being finally decrypted. This second round of mixing ensures that the privacy of users can not be compromised. A cheating server in the first round of mixing may learn at most the relationship between a double-encrypted ciphertext and a single-encrypted ciphertext, which does not help to find the corresponding plaintext after the second round of mixing.

In the rest of this section, we describe these two building blocks in greater detail.

4.1 Proof of product with checksum.

Consider a mix server who receives as inputs n ElGamal ciphertexts (G_i, M_i), and outputs a permuted re-randomization of these, namely a permutation of the list of $(G'_i, M'_i) = (G_i g^{r_i}, M_i y^{r_i})$. Our key idea is to let the mix server prove that its operations are *product preserving*, i.e. that the product of the plaintexts corresponding to the input ciphertexts (G_i, M_i) equals the product of the plaintexts corresponding to the output ciphertexts (G'_i, M'_i). The following property of the ElGamal encryption scheme makes this possible:

Proposition 1. *(Multiplicative homomorphism of ElGamal) Let* (G_1, M_1) *and* (G_2, M_2) *be ElGamal encryptions of plaintexts* P_1 *and* P_2*. Then* (G_1G_2, M_1M_2) *is an ElGamal encryption of the product* P_1P_2*. We call* (G_1G_2, M_1M_2) *the "product" of* (G_1, M_1) *and* (G_2, M_2)*.*

Proposition 1 shows that any verifier can compute an ElGamal encryption (G, M) of $\prod m_i$, and an ElGamal encryption (G', M') of $\prod m'_i$, where m_i (resp. m'_i) is the plaintext corresponding to (G_i, M_i) (resp., (G'_i, M'_i)). To prove that its operations are *product preserving*, the mix server need only prove that $\log_g(G'/G) = log_y(M'/M)$. As we saw in section 3.1, this implies that $\prod m_i = \prod m'_i$.

The need for a checksum.
The product equality $\prod m_i = \prod m'_i$ clearly does not imply that the sets $\{m_i\}_{i=1}^n$ and $\{m'_i\}_{i=1}^n$ are equal. In other words, the property of being product-preserving

does not by itself guarantee that a mix net operates correctly. Our approach is to restrict the plaintexts m_i (and therefore also m'_i) to a particular format, in such a way that it becomes infeasible for a dishonest mix server to find a set $\{m'_i\} \neq \{m_i\}$ such that $\prod m_i = \prod m'_i$ *and* all the elements m'_i are of the required format. We propose to define this special format by adding a cryptographic checksum to the plaintext, drawing on the techniques of [JJ01]. This is done as follows.

Users format their inputs to the mix net as an ElGamal encryption of a plaintext m and an ElGamal encryption of $h(m)$, where $h : \{0,1\}^* \to G_Q$ is a cryptographic hash function (in the proof of security, we model h has a random oracle [BR93]):

$$\Big(E_y(m,r), E_y(h(m),r')\Big)$$

Each input to the mix now consists of a *pair* of ElGamal ciphertexts. The mix re-randomizes separately each of the two ElGamal ciphertexts in every pair, then outputs all the pairs in a random order. The mix must then prove that the products of the plaintexts corresponding to the first element in the pair are the same in the input and the output ($\prod m_i = \prod m'_i$) and also that the products of the plaintexts corresponding to the second element in the pair are the same in the input and the output ($\prod h(m_i) = \prod h(m'_i)$). As we shall prove in section 6, these two proofs together guarantee the set equality $\{m_i\} = \{m'_i\}$.

4.2 Double Enveloping

As we have already pointed out, a mix whose correctness was enforced only by a proof-of-product with redundancy may not detect server cheating until after the harm is done. To illustrate how users' privacy may be compromised even if all cheating servers are disqualified, we offer the following example. Assume that the first mix server is corrupt and that the input submitted by user i is $(E_y(m_i,r_i), E_y(h(m_i),r'_i))$. The corrupt first server can replace the input of user 1 by $(E_y(m_1,r_1)E_y(m_2,r_2), E_y(h(m_1),r'_1)E_y(h(m_2),r'_2))$ (recall the definition of the product of ElGamal ciphertexts in Section 4.1), and replace the input of user 2 by $(1,1,1,1)$. Such cheating will only be detected after the decryption phase. Even if the cheating server were to be disqualified and the mixing protocol restarted, the cheating server would still be able to distinguish the plaintexts submitted by users 1 and 2 from other users' plaintexts, by comparing the output of the restarted protocol with that of the first execution.

To defend against this attack, we add a second layer of encryption to the plaintext m of a user. A user whose plaintext input is m is required to submit the following triple of ciphertexts to the mix:

$$(E_y(G,r), E_y(M,r'), E_y(h(G,M),r'')),$$

where $(G,M) = (g^{\hat{r}}, my^{\hat{r}})$, and as before $h : \{0,1\}^* \to G_Q$ is a cryptographic hash function.

Thus (G, M) replaces m in the description of POP-with-checksum above. (Other double enveloping designs resulting in the same functional structure are possible. We choose this one for concreteness.) If cheating is caused by a corrupt server, we can re-randomize all the inner-layer encryptions and their order with a standard ElGamal-based re-randomization mix net, before they are finally decrypted to plaintexts. Although the adversary might be able to link some inner-layer encryptions to the input ciphertexts, he cannot link the final output plaintexts to them.

5 Exit-Poll Mix Net

Assumptions. We assume that there exists a bulletin board, which is accessible to the public, and is authenticated, tamper-proof, and resistant to denial-of-service attacks. All messages and proofs are posted on this bulletin board.

Setup. The mix servers jointly generate parameters (P, Q, g, x, y) for an ElGamal cryptosystem E. The public parameters are made public, while the private key x is shared among the mix servers in a (t, n)-threshold VSS scheme. Users are required to submit their input m_i to the mix net formatted as follows:

1. The user encrypts the input m_i to produce $E_y(m_i) = (G_i, M_i)$.
2. The user computes $H_i = h(E_y(m_i))$. As explained earlier, we model h as a random oracle in the proof of security. In practice, a publicly available hash function such as MD5 [Riv92] or SHA-1 [N95] should be used.
3. The user submits the triple $E_y(G_i), E_y(M_i), E_y(H_i)$. The mix servers check that every component belongs to G_Q, and that this input has not already been submitted. If any component is not in G_Q, the user is disqualified and the triple is discarded. If the same input has already been submitted by another user, the duplicate submission is discarded.
4. The user proves his knowledge[6] of G_i, M_i, H_i. This is important to prevent a user from re-encrypting and re-posting another user's input. This proof of knowledge should be bound to a unique mix-session identifier to achieve security across multiple invocations of the mix. Any user who fails to give the proof is disqualified, and the corresponding input is discarded.
5. We note that dishonest users may submit inputs that are not properly formatted, in the sense that the equality $H_i = h(E_y(m_i))$ does not hold. We stress that such improperly formatted inputs can *not* force our mix net to default to the slower back-up mixing. The only event that can trigger a default to the back-up mixing is cheating by one of the mix servers.

First stage: re-randomization and mixing. This step proceeds as in all re-randomization mix nets based on ElGamal. One by one, the mix servers re-randomize all the inputs and their order. (Note that the components of triples

[6] We note that for reasons of efficiency, it suffices that he proves knowledge of one of these components, and make the proof relative to the other two. This can be done (as is standard) by letting the latter two be part of the input to the random oracle that sets the challenge for the proof.

are not separated from each other during the re-randomization.) In addition, each mix net must give a proof that the product of the plaintexts of all its inputs equals the product of the plaintexts of all its outputs.

1. Each mix server reads from the bulletin board the list of triples corresponding to re-encryptions of $E_y(G_i), E_y(M_i), E_y(H_i)$ output by the previous mix server: $\{(g^{r_i}, a_i \cdot y^{r_i}), (g^{s_i}, b_i \cdot y^{s_i}), (g^{t_i}, c_i \cdot y^{t_i})\}_{i=1}^{N}$. (Note that even if some servers have cheated, the ciphertexts can still be formatted like that, provided that every component belongs to G_Q.)
2. The mix server re-randomizes the order of these triples according to a secret and random permutation. Note that it is the order of triples that is re-randomized, and that the three components $E_y(G_i)$, $E_y(M_i)$ and $E_y(H_i)$ that make up each triple remain in order.
3. The mix server then re-randomizes each component of each triple independently, and outputs the results: $\{(g^{r'_i}, a'_i \cdot y^{r'_i}), (g^{s'_i}, b'_i \cdot y^{s'_i}), (g^{t'_i}, c'_i \cdot y^{t'_i})\}_{i=1}^{N}$.
4. The mix server proves that $\prod a_i = \prod a'_i$ and $\prod b_i = \prod b'_i$ and $\prod c_i = \prod c'_i$.

Second stage: decryption of the inputs.

1. A quorum of mix servers jointly decrypt each triple of ciphertexts to produce the values G_i, M_i and H_i, using the technique we reviewed in Section 3.2.
2. All triples for which $H_i = h(G_i, M_i)$ are called *valid*.
3. Invalid triples are investigated according to the procedure described below. If the investigation proves that all invalid triples are *benign* (only users cheated), we proceed to step 4. Otherwise, the decryption is aborted and we continue with the back-up mixing.
4. A quorum of mix servers jointly decrypts the ciphertexts (G_i, M_i) for all valid triples. This successfully concludes the mixing. The final output is defined as the set of plaintexts corresponding to valid triples.

Special step: investigation of invalid triples. The investigation proceeds as follows. The mix servers must reveal the path of each invalid triple through the various permutations. For each invalid triple, starting from the last server, each server reveals which of its inputs corresponds to this triple, and how it re-randomized this triple. The cost of checking the path of an invalid triple is three exponentiations per mix server (the same cost as that incurred to run one input through the mix net). One of two things may happen:

– **Benign case (only users cheated):** if the mix servers successfully produce all such paths, the invalid triples are known to have been submitted by users. The decryption is resumed after the incorrect elements have been removed.
– **Serious case (one or more servers cheated):** if one or more mix servers fail to recreate the paths of invalid triples, these mix servers are accused of cheating and replaced, and our mix terminates without producing an output. In this case, the inputs are handed over to the back-up mixing procedure described next.

Note that when the mix servers investigate an invalid triple we assume implicitly that the successive permutations applied by mix servers define a unique path for each triple through the mix net. This is not strictly true if two or more triples encode the same inner-layer ciphertext. Indeed if two triples correspond to different outer-layer encryptions of the same inner-layer ciphertext, the (outer-layer) re-encryption of one can be passed off as a re-encryption of the other. In this case, the permutations do not strictly commit mix servers to a unique path for each triple. Observe however that this does not affect the investigation of invalid triples. A corrupt server may substitute one copy (i.e. outer-layer encoding) of an invalid triple for another, but must eventually account for all copies.

Back-up mixing. The *outer-layer* encryption of the inputs posted to the mix net is decrypted by a quorum of mix servers. The resulting set of *inner-layer* ciphertexts becomes the input to a standard re-encryption mix net based on ElGamal (using, for example, Neff's scheme described in [Nef01]). At the end of this second mixing, the ciphertexts are finally decrypted to plaintexts, which concludes the mixing.

6 Security Analysis

We start with a brief discussion of the efficiency of our scheme. The costs are as follows for a batch consisting of n inputs:

- Re-encryption and mixing: **linear** number of modular exponentiations ($6n$).
- Proof of correct mixing: **constant** number of modular exponentiations (but number of modular multiplications linear in n).
- Verification: **constant** number of modular exponentiations per server (but number of modular multiplications linear in n). The cost is also linear in the number of servers.
- Decryption: **linear** number of modular exponentiations ($(5 + 10k)n$ for k servers).

This makes our mix not only twice as fast as the next fastest mix network [Nef01], but also the only mix (among mixes with standard security guarantees) for which the costs are incurred mostly in the re-encryption and decryption phases. This is important because these two phases (unlike the proof phase) can benefit from the significant speed-up techniques developed in [Jak99]. Using addition chains, we estimate that the cost of one exponentiation is roughly equivalent to 10 multiplications, with reasonably sized batches.

We now turn to proving that our mix network offers guarantees of correctness, verifiability and privacy.

Proposition 2. *(Correctness) If all parties follow the protocol, the output of the mix net is a permuted decryption of the input.*

Since the set of plaintexts is preserved in re-randomizations, this follows from the correctness of decryption.

The verifiability of our mix net is a restricted form of universal verifiability in the sense that only the operation of the mix net on *valid* inputs (*i.e.*, the inputs that are well-formed according to our protocol) are universally verifiable. We call this restricted form of verifiability "public verifiability".

Definition 1. *(Public Verifiability) A mix net is* publicly verifiable *if there exists a polynomially bounded verifier that takes as input the transcript of the mixing posted on the bulletin board, outputs "valid" if the set of valid outputs is a permuted decryption of all valid inputs, and otherwise outputs "invalid" with overwhelming probability. Note that to prove public verifiability, we consider an adversary that can control all mix servers and all users.*

Proposition 3. *Our mix net is publicly verifiable if there exists a group G in which the discrete logarithm problem is hard.*

Proof. The proof proceeds by contradiction. We assume that one or several mix servers cheat during the execution of a mixing protocol, yet manage to produce a transcript that fools an outside verifier into believing that the mixing was done correctly. We show how to use these cheating mix servers to compute discrete logarithms in the group G. Our proof is based on the following lemma:

Lemma 1. *Let a and b be two elements of a group G of order $|G|$. For random values $r_1, \ldots, r_N$ and $s_1, \ldots, s_N$, we compute the following group elements: $h_i = a^{r_i} b^{s_i}$. Consider an adversary who on inputs $h_1, \ldots, h_N$ outputs integers $e_1, \ldots, e_N$ such that $\prod_{i=1}^N h_i^{e_i} = 1$ and at least one of the e_i's is non-zero. With probability $1 - 1/|G|$, the knowledge of these e_i's allows us to compute $\log_a b$.*

Proof. If $\sum_{i=1}^N s_i e_i \neq 0$, then we can compute $\log_a b = -(\sum_{i=1}^N r_i e_i)/(\sum_{i=1}^N s_i e_i)$. It remains to prove that $\sum_{i=1}^N s_i e_i \neq 0$ happens with probability $1-1/|G|$. Since the values r_i's are random, the knowledge of the h_i's yields no information to the adversary about the s_i's. Indeed we have $\log h_i = r_i + \log_a b s_i$. Since the distribution of r_i is uniformly random, the distribution of s_i is also uniformly random given h_i. The probability that the vector $E = (e_1, \ldots, e_N)$ chosen by the adversary is orthogonal to an unknown random $S = (s_1, \ldots, s_N)$ is $1-1/|G|$. □

Now let us turn to the proof of proposition 3. We denote the inputs to the mix network as

$$(E_y(G_1, r_1), E_y(M_1, r_1'), E_y(H_1, r_1'')), \ldots, (E_y(G_N, r_N), E_y(M_N, r_N'), E_y(H_N, r_N'')),$$

and denote the outputs of the mix network as

$$(E_y(\overline{G_1}, \overline{r_1}), E_y(\overline{M_1}, \overline{r_1'}), E_y(\overline{H_1}, \overline{r_1''})), \ldots, (E_y(\overline{G_N}, \overline{r_N}), E_y(\overline{M_N}, \overline{r_N'}), E_y(\overline{H_N}, \overline{r_N''})).$$

For cheating to escape detection, the equation

$$\prod_i H_i = \prod_i \overline{H_i} \tag{1}$$

must hold, and in addition we must have $\overline{H_i} = h(\overline{G_i}, \overline{M_i})$ for all i. Furthermore, since we restrict the notion of universal verifiability to valid inputs, we have $H_i = h(G_i, M_i)$ for all i. Equation 1 can therefore be rewritten:

$$\prod_i h(G_i, M_i) = \prod_i h(\overline{G_i}, \overline{M_i}). \tag{2}$$

Now recall that in our security proof, we model the hash function h as a random oracle. Each time a mix server queries h on a new input, we choose random values r_i and s_i and return $a^{r_i} b^{s_i}$ (we answer queries on inputs that have already been queried consistently). Since the mix server cheated, equation 2 gives us a non-trivial product relationship of the type that allows us to compute discrete logarithms in the group G according to lemma 1, and this concludes the proof. $\square$

Our mix network offers the same guarantee of privacy as all mix networks based on ElGamal re-encryptions, e.g. [Nef01].

7 Conclusions

We constructed a verifiable mix network that is extremely fast in case none of the mix servers cheat. This enables election officials to quickly announce the results in the common case when all mix servers honestly follow the mixing protocol. In case one or more of the mix servers cheat, our system detects the cheating server or servers and then redoes the mixing using one of the standard (slower) mix systems [Nef01]. We emphasize that server cheating cannot compromise user privacy; it just causes the mixing process to run slower.

Our fast verifiable mixing is achieved by using the homomorphic property of ElGamal encryption to quickly test that the product of all plaintext inputs is equal to the product of all plaintext outputs. Clearly, this simple product test is insufficient for proving correct mixing. However, we are able to prove that by adding an appropriate checksum to all inputs this product test becomes sufficient. Furthermore, we use double enveloping to ensure user privacy in case one or more mix servers cheat. We hope that our approach can be used to speed-up other secure distributed computations in case all participants honestly follow the protocol, without affecting security in case of cheating.

References

[Abe98] Universally verifiable mix-net with verification work independent of the number of mix-servers. In *Proc. of Eurocrypt '98*, pp. 437-447. Springer-Verlag, 1998. LNCS 1403.

[Abe99] M. Abe. Mix-networks on permutation networks. In *Proc. of Asiacrypt '99*, pp. 258-273, 1999. LNCS 1716.

[AH01] M. Abe and F. Hoshino. Remarks on mix-networks based on permutation networks. In *Proc. of PKC'01.*

[Bon98] D. Boneh. The Decision Diffie-Hellman Problem. In *ANTS-III*, pp. 48-63, 1998. LNCS 1423.

[BF97] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. In *Proc. of CRYPTO'97*, pp. 425-439. LNCS 1294.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proc. of CCS'93*, pp. 62-73.

[Cha80] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM*, 24(2):84-88, 1981.

[CP92] D. Chaum and T. Pedersen. Wallet databases with observers. In *Proc. of Crypto'92*, pp. 89-105. Springer-Verlag, 1993. LNCS 740.

[DF89] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Proc. of Crypto'89*, pp. 307-315. LNCS 435.

[DK00] Y. Desmedt and K. Kurosawa. How to break a practical MIX and design a new one. In *Proc. of Eurocrypt'2000*, pp. 557-572. LNCS 1807.

[Fel87] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. of the 28th IEEE Symposium on the Foundations of Computer Science*, IEEE Press, 1987, pp. 427-437.

[FMMOS02] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In Pre-proceedings of Financial Crypto'02.

[FS86] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proc. of CRYPTO'86*, 1987.

[FS01] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Proc. of Crypto '01*, pp. 368-387. Springer-Verlag, 2001. LNCS 2139.

[GGMM97] E. Gabber, P. Gibbons, Y. Matias and A. Mayer. How to make personalized Web browsing simple, secure and anonymous. In *Proc. of Financial Cryptography'97*, pp. 17-31, 1997.

[GJK+99] R. Gennaro, S. Jarecki and H. Krawczyk and T. Rabin, Secure Distributed Key Generation for Discrete-Log Based Cryptosystems, In *Proc. of Eurocrypt '99*, pp. 295-310, 1999. LNCS 1592.

[GRR98] R. Gennaro, M. Rabin and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc. ACM PODC'98*, pp. 101-111.

[HS00] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *Proc. of Eurocrypt'00*, pp. 539-556. Springer-Verlag, 2000. LNCS 1807.

[Jak98] M. Jakobsson. A practical mix. In *Proc. of Eurocrypt '98*, pp. 448-461. Springer-Verlag, 1998. LNCS 1403.

[JM98] M. Jakobsson and D. M'Raïhi. Mix-based electronic payments. In *Proc. of SAC'98*, pp. 157-173. Springer-Verlag, 1998. LNCS 1556.

[Jak99] M. Jakobsson. Flash mixing. In *Proc. of PODC '99*, pp. 83-89. ACM, 1999.

[JJ99] M. Jakobsson and A. Juels. Millimix: mixing in small batches. DIMACS Technical Report 99-33.

[JJ01] M. Jakobsson and A. Juels. An optimally robust hybrid mix network. In *Proc. of PODC'01*, pp. 284-292. ACM Press. 2001.

[JJR02] M. Jakobsson, A. Juels and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. To be presented at USENIX'02.

[KY02] A. Kiayias and M. Yung. Self-tallying elections and perfect ballot secrecy. In *Proc. of PKC'02*, pp. 141-158. LNCS 2274.

[MK00] M. Mitomo and K. Kurosawa. Attack for flash mix. In *Proc. of Asiacrypt'00*, pp. 192-204. LNCS 1976.

[Nef01] A. Neff. A verifiable secret shuffle and its application to E-Voting. In *Proc. of ACM CCS'01*, pp. 116-125. ACM Press, 2001.

[N95] NIST. Secure hash standard. Federal Information Processing Standards Publication 180-1. 1995.

[OKST97] W. Ogata, K. Kurosawa, K. Sako and K. Takatani. Fault tolerant anonymous channel. In *Proc. of ICICS '97*, pp. 440-444, 1997. LNCS 1334.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of Eurocrypt'99*, pp. 223-238. LNCS 1592.

[Ped91] T. Pedersen. A Threshold cryptosystem without a trusted party. In *Proc. of Eurocrypt'91*, pp. 522-526, 1991.

[PIK93] C. Park, K. Itoh and K. Kurosawa. Efficient anonymous channel and all/nothing election Scheme. In *Proc. of Eurocrypt '93*, pp. 248-259. Springer-Verlag, 1993. LNCS 765.

[PP89] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of mixes. In *Proc. of Eurocrypt '89*, pp. 373-381. Springer-Verlag, 1989. LNCS 434.

[Pfi94] B. Pfizmann. Breaking an efficient anonymous channel. In *Proc. of Eurocrypt'94*, pp. 339-348.

[Riv92] R. Rivest. The MD5 message-digest algorithm. IETF Network Working Group, RFC 1321, 1992.

[SK95] K. Sako and J. Kilian. Receipt-free mix-type voting scheme. In *Proc. of Eurocrypt '95*. Springer-Verlag, 1995. LNCS 921.

[Sch91] C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161-174, 1991.

[TY98] Y. Tsiounis and M. Yung. On the security of ElGamal based encryption. In *Proc. of PKC'98*.