Almost Entirely Correct Mixing With Applications to Voting

Dan Boneh *
Stanford University
dabo@stanford.edu

Philippe Golle †
Stanford University
pgolle@stanford.edu

ABSTRACT

In order to design an exceptionally efficient mix network, both asymptotically and in real terms, we develop the notion of almost entirely correct mixing, and propose a new mix network that is almost entirely correct. In our new mix, the real cost of proving correctness is orders of magnitude faster than all other mix nets. The trade-off is that our mix only guarantees "almost entirely correct" mixing, i.e it guarantees that the mix network processed correctly all inputs with high (but not overwhelming) probability. We use a new technique for verifying correctness. This new technique consists of computing the product of a random subset of the inputs to a mix server, then require the mix server to produce a subset of the outputs of equal product. Our new mix net is of particular value for electronic voting, where a guarantee of almost entirely correct mixing may well be sufficient to announce instantly the result of a large election. The correctness of the result can later be verified beyond a doubt using any one of a number of much slower proofs of perfect-correctness, without having to mix the ballots again.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption

General Terms

Security

Keywords

Mix Networks, Electronic Voting

1. INTRODUCTION

A *mix server* is the cryptographic equivalent of a hat. It takes a set of input ciphertexts and outputs related ciphertexts in a random order, in such a way that the permutation that matches input to output ciphertexts is known only to the mix server and no one else. Mix servers were originally proposed by Chaum [5] to implement an untraceable email system, and have since found a wide range of applications. They are notably being used to ensure privacy in electronic voting [22, 26, 12] and anonymous payment systems [14].

To be useful, a mix server must prove that it has correctly mixed the set of input ciphertexts. That is, a mix server must prove that the set of ciphertexts it outputs matches *exactly* the set of input ciphertexts it received. Ideally, this proof should not reveal any additional information about the relationship between inputs and outputs. If we consider the example of a mix server mixing votes after an election, the proof of correct mixing guarantees that the mix server neither lost, nor added, nor modified any vote.

Proving that the output of a mix is a permutation of the inputs without compromising the secrecy of the permutation is not easy. The first solutions to this problem were based on computationally expensive zero-knowledge proofs. Much work has since been devoted to making the proofs more efficient both asymptotically and in real terms (we review related work in the following section). However, even the fastest mix [20] is still too slow to prove correctness in real time when the number of inputs is large. For instance, we estimate that to prove that one million votes have been mixed correctly would require some 20 hours on a 1GHz PC. In effect, existing mix servers are too slow to mix non-trivial numbers of inputs in real time.

In this paper, we propose an exceptionally efficient new method for proving that the output produced by a mix network is almost entirely correct. The real cost of generating a proof of almost entirely correct mixing, measured by the number of exponentiations required, is a small constant independent of the number of inputs mixed. In practice, our new mix produces an instant proof of almost entirely correct mixing. In comparison, the fastest proofs of perfectly correct mixing require work linear in the number of inputs.

Almost entirely correct mixing means that the mix network provably processed correctly all inputs with high (but not overwhelming) probability. For example, a typical application of our mix would be to give an instant proof that the outcome of an election involving one million ballots is correct with probability 99%. This guarantee may well be enough to announce the result of the election early, while a much slower perfectly correct proof runs to validate the result beyond a doubt.

We use a new technique to verify that the output of a mix server is almost entirely correct. This new technique consists

*Supported by NSF-CAREER Award
†Supported by Stanford Graduate Fellowship

CCS'02, November 18–22, 2002, Washington, DC, USA.

of first computing the product π_S of a random subset S of the inputs of the mix server, then revealing the subset S to the mix server and requesting it to produce a subset S' of the outputs such that $\pi_{S'} = \pi_S$. Observe that an honest mix server can find S' simply by applying to S the permutation that matches mix inputs to mix outputs. On the other hand, we will show in sections 6 and 7 that the problem of finding S' such that $\pi_{S'} = \pi_S$ becomes often impossible if the set of outputs produced by the mix server is not a permutation of the inputs.

The most important application of our new proof is in large electronic elections (with one million votes or more), where it may be used to guarantee almost instantly that the output produced by the mix is almost entirely correct. This guarantee will often be enough to announce the result of the election instantly. We can use in parallel to our proof any one of a number of slower proofs of perfect-correctness without having to mix the ballots again (the voters themselves, of course, need not be involved again). What makes this possible is that our proof works with the fastest and also most common implementation of mix networks: ElGamal re-encryption mix nets (see section 2.1).

The rest of this paper is organized as follows. In section 2, we define mix networks and their properties, as well as re-encryption mix nets. In section 3, we survey existing techniques for proving the correct execution of mix nets. We introduce our mechanism for proving almost entirely correct mixing in section 4. In section 5, we propose a new mix network protocol based on a proof of almost entirely correct mixing and examine the properties of this mix net in section 6. In section 7, we prove that our mix net is almost entirely correct. We conclude in section 8.

2. MIX DEFINITIONS AND PROPERTIES

A *mix server* takes as input a set of ciphertexts and outputs in a random order re-encryptions of these ciphertexts. A re-encryption of a ciphertext is a different ciphertext that decrypts to the same plaintext. We describe later in this section the encryption scheme used by mix servers and the re-encryption process. The main property desired of a mix server is that the permutation that matches input ciphertexts to output re-encryptions should be known to no one but the mix server itself.

Mixing typically involves not one but several mix servers that operate sequentially on the same data. This is called a *mix network* or mix net. Consider a mix net that consists of n servers. Users submit encrypted inputs to the mix network. These inputs are mixed in a random order by the first mix server. The output of this first mix server is passed to the second mix server, which mixes it again and passes it to the third mix server which mixes it yet again, and so on until the output of the n-th server becomes the final output of the mix net. The relationship between inputs and outputs of a mix net is not known to any single server, but could only be learnt by a coalition of all the servers.

As is customary, we model all communication between users and mix servers, as well as among mix servers with a *bulletin board*. The bulletin board is a publicly shared piece of memory to which all participants have read access and appendive, sequential write access with authentication. We further assume that all participants (users and servers) have polynomially bounded computational resources. We consider an adversary that may statically control any number of users and up to all the mix servers minus one. (We refine and justify this adversarial model in section 3). Given these assumptions, we require the following security properties of a mix network:

- **Correctness**: the set of ciphertexts output by the mix network must "match" the set of input ciphertexts. This means that every output is a re-encryption of an input, and no two outputs are re-encryption of the same input.
- **Privacy**: no adversary can match any output of the mix network to the corresponding input with probability better than $1/n$ where n is the number of inputs. Our mixnet guarantees a slightly weaker notion of privacy as explained in section 6.
- **Robustness**: the mix network must produce a correct output irrespective of possible server faults or failures.
- **Universal verifiability**: a mix net is universally verifiable if a coalition of all users and *all* mix servers only ever succeeds in convincing an outside verifier of the correct execution of a mix network when the mix network was indeed executed correctly. Note that universal verifiability uses a stronger adversarial model than that used to define privacy and correctness, since *all* mix servers may participate in the coalition to cheat an outside verifier.
- **Efficiency**: while not a security property, low computational overhead is the holy grail of secure mix networks.

2.1 Re-encryption Mix Nets

In this section, we define re-encryption mix networks. Re-encryption mix nets were originally proposed in [22]. The particular re-encryption mix of [22] was broken in [24, 25], and later fixed by [21]. A very large number of constructions based on re-encryption mix nets have since been proposed (we will survey them briefly in the following section).

Re-encryption mix networks operate in two distinct phases. In the first phase (the mixing phase), the input ciphertexts are shuffled and re-encrypted. In the second phase (the decryption phase), the mixed ciphertexts that are the output of the first phase are decrypted. The servers that perform the mixing in the first phase, and the servers that perform the decryption in the second phase need not be the same, although they *can* be the same. We treat them separately in order to define our adversarial model more clearly. We call the first group *mixing servers* and the second group *decryption servers*.

DEFINITION 2.1. *(Adversarial model) We consider an adversary that may statically control any number of users, up to all the mixing servers minus one, and up to a minority of decryption servers.*

This static adversarial model is commonly considered, but it will be particularly appropriate for the mix net we propose. Since our mixing is so fast, it is not practically important to consider dynamic adversaries. The same may not be true for slower mixes, making such schemes possibly weaker or more complicated. As we shall see, the involvement of decryption servers is very much more limited than that of

Scheme	Computational Cost				Trade-off
	Re-encryption	Proof		Decryption	
Cut and Choose ZK [26, 21]	$2n$	$642nk$	$= O(nk)$	$(2+4k)n$	
Pairwise Permutations [2, 16]	$2n$	$7n \log n(2k-1)$	$= O(nk)$	$(2+4k)n$	
Matrix Representation [9]	$2n$	$18n(2k-1)$	$= O(nk)$	$(2+4k)n$	
Polynomial Scheme [20]	$2n$	$8n(2k-1)$	$= O(nk)$	$(2+4k)n$	
Randomized Partial Checking [18]	$2n$	$n/2(2k-1)$	$= O(nk)$	$(2+4k)n$	Privacy
Optimistic Mixing [11]	$6n$	$6+12k$	$= O(k)$	$(5+10k)n$	
Proof-of-Subproduct [this paper]	$2n$	$2\alpha(2k-1)$	$= O(k)$	$(2+4k)n$	Correctness

Figure 1: Real cost per server (for a total of k servers) of mixing n items using different mixes. The cost is measured in terms of the number of exponentiations. Note that α is the security parameter of our scheme (e.g. $1 \leq \alpha \leq 5$)

mixing servers. Since this makes decryption servers easier to recruit than mixing servers, it is appropriate that our adversarial model requires a majority of honest decryption servers but only a single honest mixing server.

For concreteness, we base our presentation of re-encryption mix nets on an ElGamal implementation. ElGamal is a probabilistic public-key cryptosystem. The public parameters are a multiplicative group $\mathbb{G}$ of prime order q (either a subgroup of $\mathbb{Z}_p^*$ or an elliptic curve over $\mathbb{F}_p$), a generator g of the group $\mathbb{G}$, and an element $y = g^x$. The private key consists of the value x. ElGamal is semantically secure [27] under the assumption that the Decisional Diffie Hellman (DDH) problem is hard in $\mathbb{G}$. To encrypt a plaintext $m \in \mathbb{G}$, a user chooses a random $r \in \mathbb{Z}_q$ and produces the following ciphertext: (g^r, my^r). Note that a ciphertext is a pair of two elements of $\mathbb{G}$. To decrypt a ciphertext (a, b), a user computes a^x/b which yields the plaintext m since $(g^r)^x = y^r$.

The ElGamal cryptosystem makes re-encryption of ciphertexts possible with knowledge of only the public parameters. To re-encrypt a ciphertext $c = (g^r, my^r)$, one only needs to choose a new random value $r' \in \mathbb{Z}_q$ and compute $c' = (g^r \cdot g^{r'}, my^r \cdot y^{r'})$. It is easy to convince oneself that the ciphertexts c and c' decrypt to the same plaintext m. Furthermore, one cannot test if c' is a re-encryption of c if the DDH problem is hard in $\mathbb{G}$.

DEFINITION 2.2. *Re-encryption mix networks (ElGamal implementation)*

- **Key generation***: all decryption servers jointly generate the parameters (q, g, x, y) of an ElGamal cryptosystem in a group $\mathbb{G}$ of order q generated by g, using for example the threshold key generation protocol of Pedersen [23, 10]. The public parameters q, g, y are made public, while the private key x is shared among the decryption servers in a (t, k) secret sharing scheme.*

- **Submission of inputs***: users submit to the mix net ElGamal encrypted inputs (g^r, my^r) using the parameters generated above. Users are also required to submit a proof of knowledge for the corresponding plaintext m. This can be done for example by proving knowledge of r with respect to g^r, using my^r as input to a random oracle (see [13, 27]).*

- **Mixing phase***: mixing server M_i receives as input the set of ElGamal ciphertexts output by mixing server M_{i-1}. Server M_i permutes and re-randomizes (i.e. re-encrypts) all these ciphertexts, and outputs a new set of ciphertexts, which is then passed to M_{i+1}. Server M_i must also provide a proof of correct execution (this will be discussed in much detail below).*

- **Decryption phase***: a quorum of decryption servers jointly perform a threshold decryption of the final output, and provide a zero-knowledge proof of correctness for decryption.*

3. RELATED WORK

The main difficulty of re-encryption mixnets lies in designing computationally efficient ways for mix servers to *prove* that they mixed and re-encrypted their inputs correctly in the mixing phase. We survey here some techniques that are representative of the progress made and compare the efficiency of these techniques in Figure 1. The table in Figure 1 compares the real cost per server (for a total of k servers) of mixing n items. The cost is expressed as the number of exponentiations required to re-encrypt the inputs, verify correctness and decrypt the outputs. We do not take into account the cost of operations such as additions and multiplications that are much faster to perform than exponentiations. Where applicable, the table also mentions what trade-off is made for efficiency.

The first methods to prove the correctness of the mixing were based on cut-and-choose zero-knowledge proofs [26, 21, 1]. Though much work went into designing efficient customized zero-knowledge proofs, these schemes remain computationally expensive. To make the proofs more efficient, an approach proposed independently by Millimix [16] and MIP-2 [2, 3] is to decompose a permutation on n elements into $n \log n$ pairwise permutations called comparitors. The mix server then proves correct execution of all the comparitors one by one, which can be done efficiently with a variant of the Chaum-Pedersen [6] protocol that proves equality of discrete logarithms. The schemes recently proposed by Furukawa and Sako [9], and Neff [20] offer yet more efficient proofs of correct mixing.

We now compare in more detail our new mix net to the two schemes to which it is closest: Randomized Partial Checking [18] and Optimistic Mixing [11]. Randomized Partial Checking (RPC) trades-off some privacy for more efficiency: correctness is verified by asking each mix server to reveal a randomly selected fraction of its input/output relations. This guarantees with high probability that all but an exponentially small number of inputs were processed correctly by

the mix server. On the downside, RPC offers a weaker guarantee of privacy. Since each mix server reveals a fraction of its input/output relations, privacy becomes a *global* property of the mix network: a *majority* of honest mix servers is required to ensure privacy, rather than a *single* mix server. To achieve the same confidence in privacy, RPC requires the involvement of many more servers than the schemes we have previously surveyed.

While our mix net bears some resemblance to RPC, it exploits mostly a different trade-off. RPC trades off mostly privacy (and some correctness) for efficiency, while our scheme trades off mostly correctness (and a little privacy) for efficiency. Contrary to RPC, a mix server in our scheme does not reveal individual relationships between inputs and outputs, but only the global relationship between a large subset of the inputs and a large subset of the outputs. Like RPC, our mix offers perfect privacy if there is an honest majority of mix servers. But unlike RPC, our mix preserves some privacy even when there is only a single honest mix server. In the case of a single honest mix server, as we show in section 6, every input is hidden among $n/2^{\alpha}$ outputs on average, where n is the total number of inputs and α is the security parameter (e.g. $\alpha = 4$). On the downside, our mixnet does not guarantee perfect correctness but must rely on the parallel execution of a slower verification protocol to guarantee perfect correctness.

Finally, a proof technique similar to ours is used in [11] to build a mix network with different properties. The proof of correctness in [11] consists of a proof that the product of *all* the inputs equals the product of *all* the outputs of the mix server. This, combined with redundancy checks in the inputs, guarantees perfect correctness. As in our scheme, the cost of the proof is independent of the number of inputs mixed. But on the downside, the redundancy checks in the inputs (which guarantee perfect correctness) result in a cost of mixing and decrypting that is more than twice as high as in all other re-encryption mix networks.

To summarize, our mix has the lowest total computational overhead for mixing n inputs. In particular the number of exponentiations required to prove correct mixing is a constant independent of the number of inputs. Our scheme guarantees only almost entirely correct mixing: any error in the output is detected with probability 99% whereas perfectly correct mixes provide standard guarantees of correctness of $1 - 2^{80}$. Our scheme also trades off a little privacy (see section 6).

4. PROOF OF ALMOST ENTIRELY CORRECT MIXING

To illustrate the key idea of our proof of almost entirely correct mixing, we introduce first the following simple problem. Consider a prover who is committed to two sets of n elements: $m_1, \ldots, m_n \in \mathbb{G}$ and $m'_1, \ldots, m'_n \in \mathbb{G}$. This prover wants to convince a verifier that there exists a permutation φ on n elements such that for all i, $m'_i = m_{\varphi(i)}$. In other words, the prover must convince the verifier that the set $\{m'_i\}_{i=1}^n$ is a permutation of the set $\{m_i\}_{i=1}^n$. In addition, the prover must reveal no information about the sets $\{m_i\}_{i=1}^n$ and $\{m'_i\}_{i=1}^n$ and as little information as possible about the permutation φ. We assume that the prover is computationally bounded.

We propose the following approach:

1. The verifier chooses a random subset of indices $S \subset \{1, \ldots, n\}$ (note that $|S| \approx n/2$ with high probability).
2. The prover reveals to the verifier the set $S' = \varphi(S)$ defined as $\varphi(S) = \{\varphi(s) | s \in S\}$. The verifier checks that $|S| = |S'|$.
3. The prover shows that $\prod_S m_i = \prod_{S'} m'_i$. We assume that this can be done without revealing anything about the m_i or m'_i. The verifier is satisfied if this product equality holds.

There are a few important observations to make about this approach:

- A prover who knows a permutation φ such that for all i, $m'_i = m_{\varphi(i)}$ will always trivially succeed in steps 2 and 3.
- Consider a malicious prover who does not have a permutation φ such that for all i, $m'_i = m_{\varphi(i)}$. This prover must find a set S' such that $|S| = |S'|$ and $\prod_S m_i = \prod_{S'} m'_i$. We prove in section 7 that if the set $\{m'_i\}$ is not a permutation of the set $\{m_i\}$, then the probability that the prover can find a set S' with the desired properties in polynomial time is at most 5/8, or else the discrete logarithm problem can be solved in polynomial time in $\mathbb{G}$.
- The proof leaks a little bit of information about the permutation φ. Given S and $S' = \varphi(S)$, the verifier knows that $\varphi(i) \in S'$ if and only if $i \in S$. This is acceptable for mix network applications, considering the number of inputs m_i is typically large, so that being hidden among about half the outputs is sufficient.

Applications to ElGamal re-encryption mixes. ElGamal re-encryption mix networks were defined in section 2.1. Recall that the inputs to a mix server are ElGamal ciphertexts. The mix server mixes these inputs, re-encrypts them, and outputs a new set of ElGamal ciphertexts. Following the example above, we propose to verify that the inputs were mixed correctly by first computing the product of a randomly selected subset S of the inputs, then giving S to the mix server and asking it to produce a subset S' of the outputs whose product is the same. To make this proof technique work with ElGamal ciphertexts, we need the following two propositions:

PROPOSITION 4.1. *(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 $(g_1 g_2, m_1 m_2)$ is an ElGamal encryption of the product $P_1 P_2$.*

PROPOSITION 4.2. *(Chaum-Pedersen protocol [6]) Consider a prover who knows values (g, x, h, y) in $\mathbb{Z}_p$ and also knows $\log_g x$ and $\log_h y$. The Chaum-Pedersen protocol allows this prover to convince a verifier that $\log_g x = \log_h y$ without revealing anything about these discrete logarithms. A trivial application of the Chaum-Pedersen protocol is to prove that two ElGamal ciphertexts are re-encryption of the same plaintext.*

Consider a mix server who receives as inputs n ElGamal ciphertexts $(g^{r_i}, m_i \cdot y^{r_i})$, and outputs n ElGamal ciphertexts $(g^{r'_i}, m'_i \cdot y^{r'_i})$. Proposition 4.1 shows that any verifier

can compute an ElGamal encryption (g, m) of $\prod m_i$, and an ElGamal encryption (g', m') of $\prod m'_i$. With Proposition 4.2, the mix server can then prove that $\prod m_i = \prod m'_i$ by giving a zero-knowledge proof that $\log(g'/g) = log_y(m'/m)$.

5. OUR NEW MIX NET

In this section, we integrate our proof of almost entirely correct mixing in the design of the ElGamal re-encryption mix network and give a detailed description of the resulting mix network protocol.

Setup. The decryption servers jointly generate the parameters (q, g, x, y) of an ElGamal cryptosystem in a group $\mathbb{G}$ of prime order q generated by g. The private key x such that $y = g^x$ is shared among all decryption servers in a (t, k) secret sharing scheme. This may be done using for example the (t, k)-threshold key generation protocol of Pedersen. This setup step is executed only once. After that, the same parameters can be used to mix any number of input batches. The parameters of the ElGamal cryptosystem need only be generated anew if new servers join the mix or existing servers leave the mix.

Submission of inputs.

- The servers publish the public ElGamal parameters that were generated in the setup phase.
- Users submit their inputs to the mix net encrypted with ElGamal. Let m_i be the input of user U_i. For simplicity, we assume that $m_i \in \mathbb{G}$. User U_i encrypts m_i and posts the resulting ciphertext $(g^r, m_i \cdot y^r)$ to the mix net's bulletin board. Users must also prove knowledge of m_i (see section 2.1)
- The mix servers agree on a security parameter $\alpha > 0$, where α is a small integer (say, $\alpha \leq 5$). Higher values of α provide a stronger guarantee of correct mixing but offer less privacy to the users. We examine this trade-off in detail in the next section.

Re-randomization and Mixing.
The first mix server reads users' input ciphertexts from the bulletin board, re-randomizes the ciphertexts, and writes them back to the bulletin board in random order. One by one all other mix servers perform the same operation. The output written to the bulletin board by one mix server becomes the input to the next mix server, until each server has performed the following mix step exactly once:

1. Mix server M_j reads as inputs n ElGamal ciphertexts $C_i = (g^{r_i}, m_i \cdot y^{r_i})$ from the bulletin board.
2. M_j re-randomizes these ciphertexts to produce $C'_i = (g^{r'_i}, m'_i \cdot y^{r'_i})$
3. M_j outputs these new ciphertexts to the bulletin board in random order: $C'_{\varphi_j(i)}$, where φ_j is a random permutation on n elements chosen by mix server M_j. Mix server M_j is required to remember the permutation φ_j and the re-randomization factors r'_i until the verification step which we describe next is complete. The permutation φ_j and the re-randomization factors should of course be kept secret.

Verification.
Mix servers are not allowed to abort at any time during the verification. A mix server that does abort is accused of cheating.

Before verification starts, all servers jointly generate a random string r which will be used to generate random challenges. The string r is generated as follows. Each mix server M_j selects a random string r_j and commits to r_j using a non-malleable commitment scheme [8]. After all commitments are received, they are opened. The random string r is computed as $r = \oplus_j r_j$.

Next, each server in turn must prove that it re-randomized and mixed the ciphertexts correctly. The following 6 steps are repeated individually for each server. The verification step for server M_j proceeds as follows. As above, let $C_i = (g^{r_i}, m_i \cdot y^{r_i})$ for $1 \leq i \leq n$ denote the input ciphertexts received by server M_j. (We omit the subscript j in the notation of C_i for clarity). Let $C'_{\varphi_j(i)}$ be the set of outputs, where $C'_i = (g^{r'_i}, m'_i \cdot y^{r'_i})$.

1. We first verify that all C'_i are properly formatted, i.e. every C'_i consists of a pair $(s, t) \in \mathbb{G}^2$. Observe that this can be done efficiently. If $\mathbb{G}$ is a subgroup of $\mathbb{Z}^*_p$, the computational cost to verify that an element belongs to $\mathbb{G}$ is one exponentiation, but the verification for all C'_i can be batched (see [4]), resulting in a cost of a single exponentiation to verify all C'_i. If $\mathbb{G}$ is the group of points of an elliptic curve over $\mathbb{F}_p$ of prime order q, the computational cost to verify that a point is on the curve is one squaring and one cubing.
2. Using the Chaum-Pedersen protocol (proposition 4.2), mix server M_j proves that $\prod_{i=1}^n m_i = \prod_{i=1}^n m'_i$.
3. All mix servers collaborate to generate α sets $S_1, \ldots, S_\alpha$, where each set S_i is a subset of $\{1, \ldots, n\}$. The sets S_i are generated independently of one another in the following manner. Every index $1 \leq k \leq n$ is included in S_i independently at random with probability $1/2$. The randomness is derived from the random string r generated jointly by all servers at the beginning of the verification step. We examine in more detail how to generate the subsets S_i at the end of this section.
4. The sets $S_1, \ldots, S_\alpha$ are given to mix server M_j.
5. Mix server M_j must produce α subsets $S'_1, \ldots, S'_\alpha$ of $\{1, \ldots, n\}$ such that for all $1 \leq i \leq \alpha$ $|S_i| = |S'_i|$ and $\prod_{k \in S_i} m_k = \prod_{k \in S'_i} m'_k$. This product equality is proved using the Chaum-Pedersen protocol.
6. If the mix server fails in step 5, it is accused of cheating. The *decryption* servers are then called upon to inspect the transcript of the verification (steps 3, 4 and 5) on the bulletin board. If cheating is confirmed by the decryption servers, the cheating mix server is banned from any future mixing. In this case, the remaining honest servers restart the whole mixing from the beginning using the original ciphertext inputs posted by users to the bulletin board.

Decryption.
The mix network proceeds to the decryption step only if the verification step did not expose any cheating servers. A quorum of decryption servers jointly performs a threshold

decryption of the final output ciphertexts, and provides a zero-knowledge proof of correctness for decryption.

Perfectly-correct proof. If a proof that the mix net operated perfectly correctly is required, we may run a slower perfectly-correct verification step, such as for example that proposed by Neff [20]. The cost of this additional verification is not included in the analysis of our new mix net.

This completes the description of our new mix net. We end this section with a description of how to generate the sets $S_1, \ldots, S_\alpha$ in step 3 of the verification phase.

Optimization for the generation of challenges. Recall that we denote by r the randomness jointly generated by all the servers before the verification started. Let $h : \{0,1\}^* \rightarrow \{0,1\}^{160}$ be a hash function. In our security analysis, we model h as a random oracle. Let B be the content of the bulletin board just before the sets S_i are generated (i.e. everything that has been posted to the bulletin board up to that point). We use the master randomness r and the string B together with the hash function h to generate the sets $S_1, \ldots, S_\alpha$ using the following rule: index $k \in \{1, \ldots, n\}$ is included in S_i for mix server M_j if and only if the least significant bit of $h(r||B||j||i||k)$ is 1, where $||$ denotes string concatenation.

6. PROPERTIES

In this section, we examine the properties of our new mix net in terms of soundness, efficiency, robustness, privacy, correctness and finally universal verifiability.

6.1 Soundness

PROPOSITION 6.1. **(Soundness)** *Our mix net is sound, in the sense that a server who does not deviate from the protocol cannot fail the verification step.*

PROOF. A mix net who submits $S' = \varphi_j(S)$ can not fail the verification test. Recall from proposition 2.1 that our adversarial model allows the adversary to control at most all but one of the mix servers, and up to a minority of decryption servers. The involvement of the decryption servers when cheating is alleged (step 6 of the verification) guarantees soundness since a majority of them is honest. (Were it not for decryption servers, a majority of cheating mix servers could evict a minority of honest mix servers.) □

6.2 Efficiency

PROPOSITION 6.2. **(Efficiency)** *The cost of* mixing n *items is* $2n$ *exponentiations per mix server. The cost of* proving *that the mixing is almost entirely correct is* $2\alpha(2k-1)$ *exponentiations per mix server and the cost of decrypting* n *outputs is* $(2+4k)n$*, where* k *is the total number of mix servers.*

As discussed in section 3, our mix has the lowest total computational overhead to mix n inputs. In particular the number of exponentiations required to prove that mixing has been done correctly is a constant independent of the number of inputs.

6.3 Robustness

PROPOSITION 6.3. **(Robustness)** *Like any re-encryption mix network, our construction produces an output as long as a quorum of decryption servers is available to proceed with the decryption phase.*

6.4 Privacy

PROPOSITION 6.4. **(Privacy)** *Every input is hidden among* $n/2^\alpha$ *outputs on average.*

PROOF. In the verification step, each mix server must reveal the image by his secret permutation of α sets $S_1, \ldots, S_\alpha$, each of size on average $n/2$. Every input belongs either to S_i or to the complement of S_i, and thus the corresponding output belongs either to the image of S_i or to the image of the complement of S_i. The intersection of the images of α sets S_i (or their complements) is on average of size $n/2^\alpha$. □

This is the minimum privacy guaranteed by our mix network, given that we consider an adversarial model in which all but one of the mix servers may be controlled by the adversary. If we adopt a weaker adversarial model and assume that a majority of mix servers are honest, we can adapt the techniques of [18] to our mix network to guarantee perfect privacy for all the inputs with overwhelming probability.

6.5 Correctness

PROPOSITION 6.5. **(Almost Entirely Correct Mixing)** *If the set of outputs produced by a mix server is not a permutation of the inputs, then cheating will be detected with probability* $1 - (5/8)^\alpha$*, or the discrete logarithm problem in* $\mathbb{G}$ *can be solved in polynomial time.*

The proof of almost entirely correct mixing is fairly involved and is given in the next section. Let us consider a concrete example. Consider an election with $160,000$ ballots. A security parameter $\alpha = 6$ guarantees that every individual ballot is hidden among $2,500$ others. By proposition 6.5, the probability that the output set computed by the mix network is a permutation of the inputs is more than 94%.

6.6 Universal Verifiability

Our mix offers no guarantee of universal verifiability. We have already noted that a slower proof of perfect correctness should be executed in parallel with our proof and we assume that universal verifiability, if required, will come from that slower proof. We note that while it is important that the results of the election be available instantly (with our proof), it is acceptable to wait longer (a day) for a proof of universal verifiability (with another, slower proof).

7. PROOF OF CORRECTNESS (PROPOSITION 6.5)

In this section we prove Proposition 6.5 (almost entirely correct mixing). Throughout this section, we let $\mathbb{G}$ be a group of prime order q and $\mathbb{Z}_q = \{0, \ldots, q-1\}$.

THEOREM 7.1. *Let* $0 < \epsilon < \frac{1}{2}$ *be some constant. If the set of outputs produced by a mix server is not a permutation of*

the inputs, then a single challenge in step 5 *of the verification protocol exposes cheating with probability at least* $\frac{3}{8}-\epsilon$*, or the discrete logarithm problem in* $\mathbb{G}$ *can be solved in polynomial time.*

Before proving the theorem, we need the following simple fact:

LEMMA 7.2. *Let* $u_1, \ldots, u_{n+1}$ *and* $v_1, \ldots, v_{n+1}$ *be vectors in* $\mathbb{Z}_q^n$*. Suppose there is no* n*-by-*n *matrix* $M \in GL_n(\mathbb{Z}_q)$ *such that* $u_i = M \cdot v_i$ *for all* $i = 1, \ldots, n+1$*. Then there is a polynomial time algorithm that finds* $n+1$ *elements* $c_1, \ldots, c_{n+1} \in \mathbb{Z}_q$ *such that* $\sum_{i=1}^{n+1} c_i v_i = 0$ *but* $\sum_{i=1}^{n+1} c_i u_i \neq 0$*.*

PROOF. By relabelling the vectors $v_1, \ldots, v_{n+1}$ as required, we may assume that $V' = \{v_1, \ldots, v_k\}$ is a maximal linearly independent subset of $V = \{v_1, \ldots, v_{n+1}\}$. Let M be an n-by-n matrix such that $u_i = M \cdot v_i$ for all $v_i \in V'$. Such a matrix always exists since the vectors in V' are linearly independent. Now, for any $i > k$ the set $V' \cup \{v_i\}$ is linearly dependent and therefore we can find $c_0^{(i)}, \ldots, c_k^{(i)}$ in $\mathbb{Z}_q$ such that $c_0^{(i)} v_i + \sum_{j=1}^k c_j^{(i)} v_j = 0$. Note that $c_0^{(i)} \neq 0$. If the equality $c_0^{(i)} u_i + \sum_{j=1}^k c_j^{(i)} u_j = 0$ also holds, then $u_i = M \cdot v_i$. Therefore, by the assumption of the lemma, there exists $\ell > k$ such that $c_0^{(\ell)} v_\ell + \sum_{j=1}^k c_j^{(\ell)} v_j = 0$ but $c_0^{(\ell)} u_\ell + \sum_{j=1}^k c_j^{(\ell)} u_j \neq 0$. The algorithm works by building the vectors $(c_0^{(i)}, \ldots, c_k^{(i)})$ for $i = k+1, \ldots, n+1$ and outputting the first one satisfying the requirement of the lemma. □

We now prove Theorem 7.1. Assume that the set of outputs produced by the mix server is not a permutation of the set of inputs but cheating is detected with probability less than $\frac{3}{8} - \epsilon$. We view the mix server as a polynomial-time randomized algorithm, and construct an algorithm $\mathcal{A}$ that uses the mix server to compute discrete logarithms in $\mathbb{G}$. Algorithm $\mathcal{A}$ takes as input two values g and h in $\mathbb{G}$ and computes $\log_g h$ as follows:

1. Algorithm $\mathcal{A}$ creates ElGamal public and private keys (the public key will be used to encrypt the inputs to the mix server). Algorithm $\mathcal{A}$ keeps the private key to itself and gives the public key to the mix server. Note that $\mathcal{A}$ is emulating the decryption servers.

2. $\mathcal{A}$ creates n inputs $a_i = g^{r_i} \cdot h^{s_i} \in \mathbb{G}$ for $r_i, s_i \in \mathbb{Z}_q$ chosen independently at random. Let $A = \{a_1, \ldots, a_n\}$. Algorithm $\mathcal{A}$ submits ElGamal encryptions of the inputs $a_1, \ldots, a_n$ to the mix server.

3. The mix server produces an output set of n ElGamal ciphertexts. Algorithm $\mathcal{A}$ decrypts these ciphertexts to obtain the set $B = \{b_1, \ldots, b_n\} \subseteq \mathbb{G}$ of outputs.

4. Algorithm $\mathcal{A}$ generates a random subsets $S \subseteq A$ by including every element of A independently at random with probability half. It challenges the mix server to reveal the subset $F(S) \subseteq B$ of the outputs corresponding to S. Recall that the mix server must produce a subset $F(S) \subseteq B$ such that $|F(S)| = |S|$ and the product of the elements of $F(S)$ equals the product of the elements of S. If the mix server does not reply to the challenge, algorithm $\mathcal{A}$ rewinds it and queries it on a different random subset until the mix server produces a reply. Since the mix server answers challenges with probability at least $\frac{5}{8} + \epsilon$, the algorithm $\mathcal{A}$ needs to rewind the mix server less than twice on average.

5. Algorithm $\mathcal{A}$ repeats step 4 above $n+1$ times, rewinding the mix server between queries. It obtains independent random subsets $S_1, \ldots, S_{n+1}$ of the inputs and the corresponding replies $F(S_1), \ldots, F(S_{n+1})$ from the mix server.

6. Let $\chi(S) \in \{0,1\}^n$ be the characteristic vector of S for any subset S of A or B. We view $\chi(S) \in \{0,1\}^n$ as a vector in $\mathbb{Z}_q^n$. If there exists an n-by-n matrix $M \in GL_n(\mathbb{Z}_q)$ such that for all $i = 1, \ldots, n+1$ the equality $\chi(S_i) = M \cdot \chi(F(S_i))$ holds, then algorithm $\mathcal{A}$ reports failure.

7. Otherwise, there is no matrix $M \in GL_n(\mathbb{Z}_q)$ such that $\chi(S_i) = M \cdot \chi(F(S_i))$ holds for all $i = 1, \ldots, n+1$. By Lemma 7.2, we can then find in polynomial time $c_1, \ldots, c_{n+1} \in \mathbb{Z}_q$ such that:

$$\sum_{i=1}^{n+1} c_i \chi(F(S_i)) = (0, \ldots, 0) \bmod q$$

But

$$\sum_{i=1}^{n+1} c_i \chi(S_i) \neq (0, \ldots, 0) \bmod q$$

Let $(e_1, \ldots, e_n) = \sum_{i=1}^{n+1} c_i \chi(S_i) \in \mathbb{Z}_q^n$. By definition of the mix server, we know that $\prod_{a_j \in S_i} a_j = \prod_{b_j \in F(S_i)} b_j$ for all $i = 1, \ldots, n+1$. By multiplying these $n+1$ relations, we get

$$\prod_{j=1}^{n} (a_j)^{e_j} = \prod_{i=1}^{n+1} \prod_{b_j \in F(S_i)} (b_j)^{c_i} = 1.$$

Recall that $a_i = g^{r_i} \cdot h^{s_i}$. Therefore

$$\sum_{j=1}^{n} e_j (r_j + s_j (\log_g h)) = 0 \bmod q.$$

If $\sum_{j=1}^n e_j s_j = 0$ then algorithm $\mathcal{A}$ reports failure. This happens with probability at most $1/q$. Otherwise $\mathcal{A}$ outputs

$$\log_g h = -\Big(\sum_{j=1}^{n} e_j r_j\Big) / \Big(\sum_{j=1}^{n} e_j s_j\Big).$$

PROPOSITION 7.3. *Suppose the mix server produces outputs which are not a permutation of the inputs, yet manages to reply to challenges with probability greater than* $\frac{5}{8} + \epsilon$*. Then algorithm* $\mathcal{A}$ *succeeds in computing discrete logarithm with probability at least* $\epsilon^2/128 - 1/q$*.*

PROOF. By assumption, the mix server correctly answers a query with probability at least $\frac{5}{8} + \epsilon$, where the probability is taken over all 2^n possible queries and the random bits used by the mix server. By a standard counting argument, if we randomly fix the mix server's random bits, then with probability at least $\epsilon/2$ the mix server answers correctly at least $\frac{5}{8} + \frac{13}{16}\epsilon$ of all 2^n queries.

Once we fix the mix server's random bits, the set of queries that the mix server answers correctly is a well defined subset $\mathbb{S} \subseteq \{0,1\}^n$. Furthermore, we can view the mix server as a deterministic function $F : \mathbb{S} \to \{0,1\}^n$ mapping subsets of inputs to subsets of outputs. We know that $|\mathbb{S}| > 2^n(\frac{5}{8}+\frac{13}{16}\epsilon)$ with probability at least $\epsilon/2$. Assume for the rest of the proof that $|\mathbb{S}| > 2^n(\frac{5}{8}+\frac{13}{16}\epsilon)$.

For $i = 1,\ldots,n$, let $e_i \in \{0,1\}^n$ be the i-th unit vector (i.e. zeroes everywhere and a one in the i-th coordinate). The following lemma will be used several times:

LEMMA 7.4. *Suppose that $|\mathbb{S}| > 2^n/2$. Let $i \in \{1,\ldots,n\}$. There exist $v_0, v_1 \in \mathbb{S}$ such that $v_1 = v_0 + e_i$.*

PROOF. Let U be the subset of $\{0,1\}^n$ containing all vectors with a one in position i. Let $\mathbb{S}_0 = \mathbb{S} \cap \bar{U}$ and let $\mathbb{S}_1 = \mathbb{S} \cap U$. We define $f : \mathbb{S}_0 \to U$ to be the map that sends v_0 to $v_0 + e_i$. Since $|f(\mathbb{S}_0)| + |\mathbb{S}_1| = |\mathbb{S}| > |U|$, there exist $v_1 \in f(\mathbb{S}_0) \cap \mathbb{S}_1$. Then $(v_1 - e_i, v_1)$ is the pair of desired vectors. □

Next, we bound the probability that algorithm $\mathcal{A}$ aborts in step 6. We separate our analysis of the bound in two cases, depending on whether $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$ or not. We prove first that $\mathbb{S}$ must span all of $\mathbb{Z}_q^n$ when $|\mathbb{S}| > 2^n/2$.

LEMMA 7.5. *If $|\mathbb{S}| > (2^n)/2$ then the vectors in $\mathbb{S}$ span all of $\mathbb{Z}_q^n$.*

PROOF. Let $i \in \{1,\ldots,n\}$. By the previous lemma, there exist $v_0, v_1 \in \mathbb{S}$ such that $e_i = v_1 - v_0$. Therefore $e_1,\ldots,e_n$ are spanned by $\mathbb{S}$ and so $\mathbb{S}$ spans $\mathbb{Z}_q^n$. □

LEMMA 7.6. *Let $\epsilon \in [0, \frac{1}{2}]$ and $\mathbb{S} \subseteq \{0,1\}^n$. If $|\mathbb{S}| > 2^n(\frac{1}{2}+\epsilon)$, then n random and independent vectors from $\mathbb{S}$ will span $\mathbb{Z}_q^n$ with probability at least $\epsilon/4$.*

PROOF. Let $v_1,\ldots,v_c \in \mathbb{Z}_q^n$ be linearly independent vectors where $v_1,\ldots,v_c \in \{0,1\}^n$. We know that the linear space spanned by $v_1,\ldots,v_c$ contains at most 2^c vectors in $\{0,1\}^n$. Therefore, if we are given c linearly independent vectors $v_1,\ldots,v_c$ as above, the probability that a random vector from $\mathbb{S}$ is linearly dependent on $v_1,\ldots,v_c$ is at most $2^c/|\mathbb{S}|$. If follows that n random vectors from $\mathbb{S}$ span all of $\mathbb{Z}_q^n$ with probability at least

$$p = \left(1 - \frac{2}{|\mathbb{S}|}\right)\left(1 - \frac{2^2}{|\mathbb{S}|}\right)\cdots\left(1 - \frac{2^{n-1}}{|\mathbb{S}|}\right)$$

In what follows, all the logarithms are taken base 2. Since $\log_2(1-x) \geq -2x$ for $x \in [0, \frac{1}{2}]$ and $|\mathbb{S}| > 2^{n-1}$, we have

$$\begin{aligned}
\log p &= \log\left(1 - \frac{2^{n-1}}{|\mathbb{S}|}\right) + \sum_{i=1}^{n-2} \log\left(1 - \frac{2^i}{|\mathbb{S}|}\right) \\
&\geq \log\left(1 - \frac{2^{n-1}}{|\mathbb{S}|}\right) - 2\sum_{i=1}^{n-2} \frac{2^i}{|\mathbb{S}|} \\
&\geq \log\left(1 - \frac{2^{n-1}}{|\mathbb{S}|}\right) - \frac{2^n}{|\mathbb{S}|}
\end{aligned}$$

Since we assume $|\mathbb{S}| > (2^n)(\frac{1}{2}+\epsilon)$, we get $p \geq \frac{\epsilon}{2+4\epsilon}$. With $\epsilon < \frac{1}{2}$, we get the desired result. □

LEMMA 7.7. *If $|\mathbb{S}| > 2^n(\frac{1}{2}+\epsilon)$ but $F(\mathbb{S})$ does not span all of $\mathbb{Z}_q^n$, then algorithm $\mathcal{A}$ does not abort in step 6 with probability at least $\epsilon/4$.*

PROOF. By Lemma 7.6, we know that with probability at least $\epsilon/4$, the first n vectors $\chi(S_1),\ldots,\chi(S_n) \in \{0,1\}^n$ will span all of $\mathbb{Z}_q^n$. But since $F(\mathbb{S})$ does not span all of $\mathbb{Z}_q^n$, the n vectors $\chi(F(S_1)),\ldots,\chi(F(S_n)) \in \{0,1\}^n$ do not span all of $\mathbb{Z}_q^n$. Therefore there is no matrix $M \in GL_n(\mathbb{Z}_q)$ that maps $\chi(F(S_i))$ to $\chi(S_i)$ for all $i = 1,\ldots,n$ and so the algorithm does not abort in step 6. □

Next, we want to bound the probability of aborting in step 6 when $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$. For that, we first need the following two lemmas which show that F cannot be a linear map on $\mathbb{S}$. Recall that a permutation matrix is a permutation of the rows of the identity matrix. Let us define the L norm of a vector $b = (b_1,\ldots,b_n) \in \{0,1\}^n$ to be $L(b) = \sum_{i=1}^n b_i$.

LEMMA 7.8. *Let $|\mathbb{S}| > (5/8)2^n$ and $F : \mathbb{S} \to \{0,1\}^n$ be a linear function such that $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$ and F preserves the L norm (i.e. $L(F(v)) = L(v)$ for all $v \in \mathbb{S}$). Then there exists a permutation matrix $P \in GL_n(\mathbb{Z}_q)$ such that $F(v) = P \cdot v$ for all $v \in \mathbb{S}$.*

PROOF. If $F : \mathbb{S} \to \{0,1\}^n$ is a linear function, we can find an n-by-n matrix P such that for all $S \in \mathbb{S}$, $F(S) = P \cdot S$. Note that the matrix P must be of full rank since $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$. For $i, j \in 1,\ldots,n$, we write $p_{i,j}$ the entry of P at row i and column j.

We start by showing that $p_{i,j} \in \{-1,0,1\}$ for all $i, j \in 1,\ldots,n$. Recall that for $i = 1,\ldots,n$ we denote the i-th unit vector by $e_i \in \{0,1\}^n$. By Lemma 7.4, we can find $v_0, v_1 \in \mathbb{S}$ such that $v_1 = v_0 + e_i$. Since $v_0, v_1 \in \mathbb{S}$, we know that $F(v_0) = Pv_0$ and $F(v_1) = Pv_1$ are in $\{0,1\}^n$. But since $v_1 = v_0 + e_i$ we have $Pe_i = Pv_1 - Pv_0$. This shows that the vector Pe_i can be expressed as the difference between two vectors in $\{0,1\}^n$ and therefore all the coordinates of Pe_i are in $\{-1,0,1\}$. But the vector Pe_i is exactly the i-th column of the matrix P. Since this argument works for all $i = 1,\ldots,n$, we have shown that $p_{i,j} \in \{-1,0,1\}$ for all $i, j \in 1,\ldots,n$.

Recall the definition of the norm L given above. We have:

$$\begin{aligned}
L(Pe_i) &= L(Pv_1 - Pv_0) = L(Pv_1) - L(Pv_0) \\
&= L(F(v_1)) - L(F(v_0)) = L(v_1) - L(v_0) = 1
\end{aligned}$$

This implies the following equality, which we will use later:

$$\sum_{j=1}^{n}\sum_{i=1}^{n} p_{i,j} = n.$$

Let us now consider row $R_i = (p_{i,1},\ldots,p_{i,n})$ of the matrix P for $i \in 1,\ldots,n$. For any $v \in \mathbb{S}$, we know that the scalar product $R_i \cdot v \in \{0,1\}$ since $F(v) = Pv \in \{0,1\}^n$. But $|\mathbb{S}| > (5/8)2^n$ by assumption. This implies that the vector R_i has at most 11 non-zero elements (either 1 or -1). Indeed, if R_i had $z > 12$ non-zero elements, it would map at most a fraction $2\binom{z/2}{z/4}/2^{z/2} < 5/8$ of vectors in $\{0,1\}^n$ to $\{0,1\}$. Up to a re-ordering of the columns of the matrix P, we can assume that the $z \leq 11$ non-zero coordinates of R_i are the first z coordinates. An exhaustive search among all $2^1 + 2^2 + \ldots + 2^{11} = 2^{12} - 2$ possibilities of all vectors that map a fraction strictly greater than $5/8$ of all vectors in $\{0,1\}^n$

to $\{0,1\}$ reveals that every row R_i of P must be of one of the following 4 types:

- Type 1: there is only 1 non-zero coordinate and its value is 1.
- Type 2: there are only 2 non-zero coordinates and their values are $1, 1$.
- Type 3: there are only 2 non-zero coordinates and their values are $1, -1$.
- Type 4: there are only 3 non-zero coordinates and their values are $1, 1, -1$.

Note that no row can have all zero coordinates since the matrix P is of full rank. Let t_1, t_2, t_3, t_4 be the number of rows of P of type $1, 2, 3$ and 4 respectively. Observe that $t_1 + t_2 + t_3 + t_4 = n$. We also know that

$$t_1 + 2t_2 + t_4 = \sum_{j=1}^{n}\sum_{i=1}^{n} p_{i,j} = n$$

and therefore $t_2 = t_3$. Now suppose $t_2 \geq 1$. The matrix P then contains at least one row v_2 of type 2 and one row v_3 of type 3. Up to symmetries and permutations on the columns of P, there are 4 distinct ways in which the non-zero coordinates of v_2 and v_3 can be arranged relative to one another:

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

All of these arrangements map at most a fraction $5/8$ of all vectors in $\{0,1\}^n$ to vectors in $\{0,1\}^n$. This means that $t_2 = t_3 = 0$.

It remains to show that $t_4 = 0$. We proceed by contradiction. Assume there is at least one row v_4 of type 4. Up to a re-ordering of the columns of P, we may assume $v_4 = (-1, 1, 1, 0, 0, \ldots, 0)$. Since $L(Pe_1) = 1$, there are at least two other rows of P that have a 1 in the first column. These cannot both be of type 1 for otherwise they would be linearly dependent and that cannot happen since P is of full rank. Therefore there is at least one other row v_4' of type 4 which has a 1 in the first column. Up to symmetries and permutations on the columns of P, there are 4 distinct ways in which the non-zero coordinates of v_4 and v_4' can be arranged relative to one another:

$$\begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \end{pmatrix}\begin{pmatrix} -1 & 1 & 1 & 0 \\ 1 & 1 & 0 & -1 \end{pmatrix}$$

$$\begin{pmatrix} -1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{pmatrix}\begin{pmatrix} -1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & -1 \end{pmatrix}$$

All of these arrangements map at most a fraction $5/8$ of all vectors in $\{0,1\}^n$ to vectors in $\{0,1\}^n$. This means that $t_4 = 0$. Therefore all the rows of P are of type 1 so that P is a permutation matrix. □

LEMMA 7.9. *Let $|\mathbb{S}| > 2^n/2$ and $F : \mathbb{S} \to \{0,1\}^n$ be a linear function such that $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$. If there exists a permutation matrix $P \in GL_n(\mathbb{Z}_q)$ such that $F(v) = Pv$ for all $v \in \mathbb{S}$, then the outputs B of the mix server are a permutation of the inputs A.*

PROOF. Consider the unit vector e_j for $j \in 1, \ldots, n$. Since P is a permutation matrix there exists a unit vector e_i such that $Pe_i = e_j$. By Lemma 7.4, we can find $v, w \in \mathbb{S}$ such that $w = v + e_i$. Let us write the coordinates of v and w as $v = (v_1, \ldots, v_n)$ and $w = (w_1, \ldots, w_n)$. We also write $F(v) = (v_1', \ldots, v_n')$ and $F(w) = (w_1', \ldots, w_n')$. We have

$$F(w) = Pw = Pv + Pe_i = F(v) + e_j.$$

This implies that

$$\prod_{k=1}^{n}(a_k)^{w_k} = a_i \prod_{k=1}^{n}(a_k)^{v_k}$$

$$\prod_{k=1}^{n}(b_k)^{w_k'} = b_j \prod_{k=1}^{n}(b_k)^{v_k'}$$

Since $v, w \in \mathbb{S}$, $\prod_{k=0}^{n}(a_k)^{v_k} = \prod_{k=1}^{n}(b_i)^{v_k'}$ and $\prod_{k=1}^{n}(a_k)^{w_k} = \prod_{k=1}^{n}(b_k)^{w_k'}$. Thus we must have $b_j = a_i$. The same reasoning holds for all $1 \leq j \leq n$, which shows that the mix server didn't cheat. □

Lemmas 7.8 and 7.9 show that when $|\mathbb{S}| > (5/8)2^n$ and $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$, then F cannot be a linear map on $\mathbb{S}$ unless the mix server did not cheat. The next lemma shows that when $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$, step 6 succeeds with non-negligible probability.

LEMMA 7.10. *If $|\mathbb{S}| > 2^n(\frac{5}{8} + \frac{13}{16}\epsilon)$ and $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$, then algorithm $\mathcal{A}$ does not abort in step 6 with probability at least $\epsilon/64$.*

PROOF. By Lemma 7.6, we know that with probability at least $1/32$, the first n vectors in step 6 of algorithm $\mathcal{A}$, $\chi(S_1), \ldots, \chi(S_n) \in \{0,1\}^n$ will span all of $\mathbb{Z}_q^n$. Let $M \in GL_n(Z_q)$ be the unique matrix that maps $\chi(F(S_i))$ to $\chi(S_i)$ for all $i = 1, \ldots, n$.

Let $T \subseteq \mathbb{S}$ be the subset of all the vectors $v \in \mathbb{S}$ such that $v = M \cdot F(v)$. By Lemmas 7.8 and 7.9, we know that $|T| \leq (5/8)2^n$. Therefore the probability that the $n+1$'st vector $\chi(S_{n+1})$ in step 6 of algorithm $\mathcal{A}$ is not in T is at least $\frac{13}{16}\epsilon$. When that happens, step 6 will not abort. The probability of not aborting in step 6 is therefore at least $(1/32)(\frac{13}{16}\epsilon) \geq \epsilon/64$ as required. □

Proof of Proposition 7.3

To summarize, we prove here the lower bound given in Proposition 7.3 on the probability that algorithm $\mathcal{A}$ succeeds in outputting $\log_g h$. Algorithm $\mathcal{A}$ only ever aborts in steps 6 and 7. We know that the probability that $\mathcal{A}$ aborts in step 7 is at most $1/q$. Furthermore, we have shown that with probability at least $\epsilon/2$, we have $|\mathbb{S}| > 2^n(\frac{5}{8} + \frac{13}{16}\epsilon)$. When that happens:

- When $F(\mathbb{S})$ does not span all of $\mathbb{Z}_q^n$, the probability that $\mathcal{A}$ does not abort in step 6 is at least $1/32$ by Lemma 7.7.
- When $F(\mathbb{S})$ spans all of $\mathbb{Z}_q^n$, the probability that $\mathcal{A}$ does not abort in step 6 is at least $\epsilon/64$ by Lemma 7.10.

It follows that the probability of success of $\mathcal{A}$ is at least $\epsilon^2/128 - 1/q$. □

8. CONCLUSION

The strongest point of our new mix network is its exceptional speed. The real cost of proving almost entirely correct mixing is orders of magnitude faster than all other mix networks. An almost entirely correct output is available instantly and can be announced long before it is confirmed by a slower perfectly correct mix network.

In practice, our new mix is of particular interest to large electronic elections (say, a million ballots or more), where a guarantee of almost entirely correct mixing may well be sufficient to announce the outcome of an election pending confirmation by a slower perfectly correct mixnet. This additional proof of perfect correctness does not require the ballots to be mixed again, and of course doesn't require any involvement from the voters.

We propose the first construction that exploits a trade-off between efficiency and correctness. An interesting direction for future work would be to study this trade-off further. In particular, it would be interesting to determine whether it is possible to build mix nets that span the entire continuum of the trade-off between efficiency and correctness.

9. ACKNOWLEDGMENTS

The second author wishes to thank Markus Jakobsson and Ari Juels for helpful conversations and comments on early versions of this paper.

10. REFERENCES

[1] M. Abe. 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.

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

[3] M. Abe. Remarks on mix-networks based on permutation networks.

[4] M. Bellare, J. Garay and T. Rabin. Batch Verification with Applications to Cryptography and Checking. In *Proc. of Eurocrypt '98*, pp. 170-182. Springer Verlag, 1998. LNCS 1380.

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

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

[7] 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.

[8] D. Dolev, C. Dwork, M. Naor. Nonmalleable Cryptography. In SIAM J. Comput. 30(2): 391-437 (2000)

[9] 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.

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

[11] P. Golle, S. Zhong, D. Boneh, M. Jakobsson and A. Juels. Optimistic Mixing for Exit-Polls. To appear in Asiacrypt 2002.

[12] 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.

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

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

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

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

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

[18] M. Jakobsson, A. Juels and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proc. of USENIX'02*.

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

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

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

[22] 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.

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

[24] 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.

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

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

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