Parallel Mixing

Philippe Golle
Palo Alto Research Center
Palo Alto, CA 94304, USA
pgolle@parc.com

Ari Juels
RSA Laboratories
Bedford, MA 01730, USA
ajuels@rsasecurity.com

ABSTRACT

Efforts to design faster synchronous mix networks have focused on reducing the computational cost of mixing per server. We propose a different approach: our re-encryption mixnet allows servers to mix inputs in parallel. The result is a dramatic reduction in overall mixing time for moderate-to-large numbers of servers. As measured in the model we describe, for n inputs and M servers our parallel re-encryption mixnet produces output in time at most $2n$ – and only around n assuming a majority of honest servers. In contrast, a traditional, sequential, synchronous re-encryption mixnet requires time Mn.

Parallel re-encryption mixnets offer security guarantees comparable to those of synchronous mixnets, and in many cases only a slightly weaker guarantee of privacy. Our proposed construction is applicable to many recently proposed re-encryption mixnets, such as those of Furukawa and Sako, Neff, Jakobsson et al., and Golle and Boneh. In practice, parallel mixnets promise a potentially substantial time saving in applications such as anonymous electronic elections.

Categories and Subject Descriptors

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

General Terms

Security

Keywords

Anonymity, Privacy, Mixnet, Permutation, Parallel execution

1. INTRODUCTION

A *mix network* or *mixnet* is a fundamental cryptographic tool for privacy enforcement invented by Chaum [4]. Mixnets can help achieve a broad range of privacy applications. Among the most commonly considered are anonymous message delivery systems (such as Mixminion [5]) and privacy-preserving electronic elections, as described recently in, e.g., [8, 16].

There is now an extensive literature on mixnet designs [4, 6, 9, 12, 11, 14, 13, 15, 19, 20, 21]. While there are many varieties of mixnets, the core operation of a mixnet is always essentially the same. A mixnet takes as input a set of n ciphertexts, re-encrypts (or decrypts) these ciphertexts, and outputs the n new ciphertexts (or plaintexts) in a randomly permuted order. In this paper, we consider exclusively re-encryption mixnets, i.e. mix networks that re-encrypt input ciphertexts and output new ciphertexts in a randomly permuted order.

Re-encryption mixing is typically followed by a decryption step, in which the final ciphertext outputs of the re-encryption mixnet are decrypted by a quorum of entities that share the decryption key. Our parallelization technique applies only to the mixing and re-encryption operations (by far the most computationally expensive), not to the final decryption step (ElGamal threshold decryption, in any case, lends itself naturally to parallelization). In what follows, we thus leave aside the final decryption step, and focus on the re-encryption and mixing operation.

We describe next the re-encryption operation in more detail. The ciphertext inputs and outputs of a re-encryption mix network have the same underlying plaintexts. Where they differ is in the *ordering* of these plaintexts. With respect to plaintexts, the outputs of a mixnet are the result of a random, secret permutation applied to the corresponding inputs. The re-encryption operation hides the correspondence between inputs and outputs (i.e. the secret permutation) from outside observers, and underlies the privacy properties of a mixnet.

To help ensure the secrecy of this permutation, the operations of a mixnet are distributed across multiple computing platforms, often generically referred to as servers. A well designed mixnet preserves secrecy even in the face of active adversarial compromise of a number of servers. Mixnets can be divided into two categories depending on how inputs are assigned to servers. In *synchronous* mixnets (also known as mix cascades), each server mixes in turn a full batch of n ciphertexts, then passes the batch on to the next server. *Asynchronous* mixnets let servers exchange and mix inputs freely, without preserving the structure of a batch.

At any given time in a conventional synchronous mixnet, all servers but one are idle. Most of the computational resources of such mixnets may in consequence be viewed as wasted (A partial exception to this is the synchronous

CCS'04, October 25-29, 2004, Washington, DC, USA.

mixnet of [1, 2], which permits some operational overlap.) Given M servers, the total running time of a synchronous mix network is $\Omega(Mn)$, as each server must in turn perform $\Omega(n)$ work to process n input elements. Asynchronous mixnets are more efficient, since they permit a more efficient distribution of inputs among servers, but they offer weaker guarantees of privacy [7] and in particular suffer from traffic analysis attacks [22, 25].

In this paper, we show how synchronous re-encryption mixnets may be parallelized so as to harness the computational power of all servers efficiently. A naive approach would be to divide the set of n inputs into λ batches each of size n/λ, and feed these smaller batches to the synchronous mixnet one after the other, keeping the pipeline of mix servers full. The obvious drawback of this approach is that it offers much weaker privacy, since every input is mixed only with n/λ others instead of being mixed among the full set of n inputs.

In contrast, we propose a parallelization technique that keeps nearly intact the strong privacy guarantees of synchronous mixnets. The result is a striking drop in the full operating time of a synchronous re-encryption mixnet from Mn to $2n$, or even to n when a majority of servers are honest. In practice, our proposal accelerates total mixing time by roughly a factor of $M/2$ or better. In cases where $M > 2$ and where communications latency is small, this can have a substantial practical impact.

In both sequential mixnets and our proposed parallel mix network, each server implements one or more local mixing operations on sets of input ciphertexts. In other words, a server applies its own secret, random permutation. (The global permutation applied by the mixnet represents the composition of permutations applied by individual servers.) We refer to a server's local permutation operation as a *mixop*.

Any robust mixnet includes a cryptographic mechanism enabling servers to prove that their mixops have been performed correctly, i.e., that the set of plaintexts corresponding to inputs is identical to that of the outputs. The literature offers many tools to accomplish this goal [1, 9, 12, 11, 19]. Our parallel constructions in this paper are agnostic to the choice of these proof techniques. In other words, we offer a way to restructure any existing re-encryption mixnet architecture so as to parallelize it. We do not propose any change to the underlying cryptographic technology. Rather, we show how to enable servers to perform many mixops on small ciphertext sets, rather than monolithic mixops on a full set of n ciphertexts. Given this focus, it is convenient to make a few ideal modelling assumptions:

1. **Ideal correctness:** We assume that server misbehavior is always detected, i.e., we assume perfect public verifiability. In other words, we abstract away the underlying cryptographic features of mixop proofs, and instead assume an ideal proof mechanism. We do not assume that a server can be detected if it fails to apply a truly random permutation in performing its mixops. We only assume full verifiability of the fact that a server has executed *some* valid permutation.

2. **No communication latency:** We assume that data may be transmitted instantaneously between any two servers. This assumption allows us to ignore the time required for communication and focus on the time required for computation. In practice, computation time is likely to dwarf communication time in many situations, such as when input sizes are fairly large. Where communication time is non-negligible, it can simply be added to our estimates of computation time.

3. **Bulletin board:** As is common in the mixnet literature, we assume use of a *bulletin board*. This is a piece of publicly accessible memory with universal read access and appendive write access. In the case where a bulletin board is maintained by servers participating in a mixnet, it may be achieved using Byzantine agreement as described in [18], for which a recent, practical construction is proposed in [3]. We consider an adversarial model in which the adversary may control a majority of servers. In this case, we must assume that the bulletin board is reliably maintained by an external entity.

4. **Linear mixop costs:** We assume that proof and re-encryption operations for mixops are linear in the number of input elements. This holds for many mixnets, as we explain below. We treat the full processing of a single element as an atomic computational operation; we refer to a single such operation as a *step*. We furthermore assume uniform computing time across servers, so that a step may also be defined as the interval of time required to process a single element.

5. **Ideal randomness source:** We assume that a publicly verifiable, ideal source of randomness is available for our mixnet. (This is required in particular for step 1 of our construction.) In practice, this may be embodied through a distributed protocol among servers, but for simplicity, we abstract this process away.

As a result of these assumptions, our construction is essentially combinatorial in nature. We may define a mixnet entirely as a schedule of mixops among servers, without treatment of any of the supporting cryptographic apparatus. Our basic measure of efficiency is the total interval of time (i.e. the total number of steps) required to run the mixnet when servers behave correctly.

Organization. We survey related work in the rest of this section. We define our model in section 2, and propose a parallel mixnet protocol in section 3. We prove security results in section 4, followed by some optimality results in section 5. We conclude in section 6.

1.1 Related Work

Parallelization is a natural optimization to pursue in a distributed system. It is indeed somewhat surprising that mix networks have not benefitted from such an approach. Both switching networks [24] and parallel sorting algorithms [17] accomplish much the same goal as a parallel mixnet, but in a non-adversarial setting. Achieving a truly random, secret permutation in an adversarial setting is rather more challenging.

In their mixnet constructions, Abe [1] and Jakobsson and Juels [14] have turned to a class of switching networks known as permutation networks. (A permutation network is a switching network that can realize any permutation of its inputs). Both of these proposals, however, have employed permutation networks in an essentially serial manner (although Abe does exploit a small degree of parallelization).

Our proposed construction in this paper may be viewed as a parallelized permutation algorithm specially adapted for the adversarial setting of a mixnet. Like other parallelized algorithms, ours involves the careful distribution and coordination of small tasks across multiple servers. In our construction, the small tasks in question are mixops on small input batches.

Our construction is applicable to any re-encryption mixnet whose costs are linear in the number of input elements. Among recent, efficient constructions, this criterion is fulfilled by the mixnets of Furukawa and Sako [9], Neff [19], Jakobsson et al. [15], and Golle and Boneh [10].

2. MODEL

Let n be the number of input ciphertexts to a mixnet, and let M be the number of mix servers. We structure our mixnet construction in terms of a sequence of synchronous *rounds*. A round is a period during which servers execute mixops without communicating. At the end of a round, servers may exchange batches of ciphertexts. We let R denote the number of rounds in a given mix network construction.

Upon input and at the beginning of each round of mixing, the mixnet globally operates on an ordered sequence of n ciphertexts. We think of this sequence as occupying a vector of n *slots*. Initially, each of the n input ciphertexts is placed in one of the n slots (let the i^{th} ciphertext occupy the i^{th} slot). The operation of a mixnet is defined by a schedule assigning each mix server to a subset of slots in a given round. The role of a server is to permute the ciphertexts among the slots it is assigned. These ciphertexts are at the same time re-encrypted to hide the permutation that the server has applied to elements in its slots.

Throughout the paper, we reserve the letter k for indexing mixing rounds ($k \in \{1, \ldots, R\}$), the letter j for indexing mix servers ($j \in \{1, \ldots, M\}$), the letter i for indexing input ciphertexts ($i \in \{1, \ldots, n\}$) and the letter s for indexing slots ($s \in \{1, \ldots, n\}$). Note that ciphertexts and slots should not be confused. At the beginning, the i^{th} ciphertext occupies the slot $s = i$. But whereas slots are fixed, ciphertexts move between slots in every round as they get permuted by the servers.

Given our ideal assumption about the correctness of mixing, i.e. our assumption that cheating servers are always caught, the notion of a *path* for an input element through the mixnet is well defined for a particular mixnet invocation. The path of an element is the ordered list of slots $s_1, \ldots, s_{R+1}$ occupied by that element in the course of mixing. The element occupies slot s_1 before the first round of mixing, s_2 in-between the first and second rounds of mixing, and finally s_{R+1} after the R^{th} round of mixing. (When two input plaintexts are identical, a path is defined in terms of the permutations selected by servers.)

The structure of a parallel mixnet is entirely defined by the assignment of slots to mix servers in each round. We define $S_k(j) \subseteq \{1, \ldots, n\}$ to be the set of slots assigned in round k to server j. A slot may never be assigned to more than one server in the same round, so that $S_k(j_0) \cap S_k(j_1) = \emptyset$ for all rounds k and all servers $j_0 \neq j_1$. Note however that the sets $S_k(1), \ldots, S_k(M)$ need not necessarily form a partition of $\{1, \ldots, n\}$ since some slots may not be assigned to any server in round k.

Definition 2.1. **(Parallel mixnet)** *The set $S = \{S_k(j)\}$ for $1 \leq j \leq M$ and $1 \leq k \leq R$ completely defines the structure of the parallel mixnet.*

In a parallel mixnet defined in this way, the computational cost for server j is $\sum_{k=1}^{R} |S_k(j)|$ steps. The total mixing time for the mix network is $\sum_{k=1}^{R} \max_j |S_k(j)|$ steps.

Example: A traditional, sequential mixnet may be defined as follows. The number of rounds is equal to the number of servers and therefore $R = M$. For $1 \leq j, k \leq M$, we let $S_k(j) = \{1, \ldots, n\}$ for $j = k$ and $S_k(j) = \emptyset$ otherwise. In other words, server j is assigned all n slots in round $k = j$ and mixes all n ciphertexts found in these slots. The computational cost for a single server is $O(n)$, while the total mixing time is $O(nM)$.

Definition 2.2. **(Mixnet isomorphism)** *Let the sets $S = \{S_k(j)\}$ and $T = \{T_k(j)\}$ for $j = 1, \ldots, M$ and $k = 1, \ldots, R$ define two R-round parallel mixnets on n elements. We say that S and T are isomorphic if there exists a permutation π of the n slots such that $S_k(j) = \pi(T_k(j))$ for all $j = 1, \ldots, M$ and $k = 1, \ldots, R$.*

A mixnet isomorphism amounts to a renaming of slots. Consider for example a 2-round mixnet on 3 inputs. Let $S_1(1) = \{1, 2\}$ and $S_1(2) = \{3\}$ and $S_2(1) = \{1\}$, $S_2(2) = \{2, 3\}$. In other words, server 1 is assigned slots 1 and 2 in the first round, and slot 1 only in the second round. Server 2 is assigned slot 3 in the first round, and slots 2 and 3 in the second round. Similarly, we define $T_1(1) = \{1, 2\}$ and $T_1(2) = \{3\}$ and $T_2(1) = \{2\}$, $T_2(2) = \{1, 3\}$. The permutation $\pi(1) = 2$, $\pi(2) = 1$ and $\pi(3) = 3$ defines an isomorphism between the parallel mixnets S and T. Observe that we define mixnet isomorphisms in terms of a renaming of slots, not a renaming of servers (in the example above, no renaming of servers maps S to T).

2.1 Anonymity

We define next the properties of parallel mixnets with respect to anonymity. To do so, we must first define our adversarial model. In our model:

– *The adversary may passively control up to $M - 1$ mix servers.* Given our ideal correctness assumption, we do not need to consider active adversarial attacks. Moreover, because we assume full public verifiability, we are able to consider an adversary capable of compromising all but one server. (Without public verifiability, we could only consider an adversary capable of corrupting a minority set of servers; an adversary controlling a majority would be able to "outvote" and thus override the input of honest servers.) Let $\mathcal{A}(M)$ denote the set of mix servers controlled by the adversary. Mix servers controlled by the adversary continue to follow the mixing protocol correctly, but the adversary learns all the permutations applied by these servers. If the adversary controls no server, we say that it is an *external* adversary. An external adversary is limited to observing the inputs and outputs produced by all the mix servers.

– *The adversary may learn a number of input/output relations.* More precisely, prior to the invocation of the mixing process the adversary can designate a set of up to $n - 2$ input elements. The adversary learns, for each such element, which slot the element starts from and which slot the element lands into after the mixing is complete. Note that these slots are well-defined: they are the starting point and end point of the path of the element. Let $\mathcal{A}(I)$ denote the subset of slots initially occupied by elements controlled by the adversary, and let $\mathcal{A}(O)$ denote the subset of slots in which these elements land after mixing is complete. We assume that there are at least two elements for which the adversary is unable to learn the initial and final slot, since that is the minimum requirement for a meaningful definition of privacy.

This assumption captures the important fact that an adversary may control players providing inputs and receiving outputs from the mixnet. For example, in a mixnet used for anonymous e-mail transmission, the adversary might control players sending and receiving a portion of the e-mail messages passing through the mixnet.

To define anonymity formally, we consider the probability $P_{\mathcal{A}}(k, s_0, s_1)$ that the ciphertext occupying slot s_1 at the end of round k is the image (through successive permutations) of the input ciphertext that occupied slot s_0 before mixing began, in the view of the adversary. In other words, the probability $P_{\mathcal{A}}(k, s_0, s_1)$ is the probability that the path of the element that started in slot s_0 goes through slot s_1 after k rounds of mixing, in the view of the adversary.

Definition 2.3. *We define the anonymity at slot s after round k with respect to adversary $\mathcal{A}$ as* $\mathsf{Anon}_k(s) = \left(\max_{s_0} P_{\mathcal{A}}(k, s_0, s)\right)^{-1}$. *We define the anonymity of the mixnet for round k with respect to adversary $\mathcal{A}$ as* $\mathsf{Anon}_k = \min_s \mathsf{Anon}_k(s)$.

The value Anon_k gives a lower bound on the size of the smallest input set into which an adversary may trace a given element in the mixnet. For example, if $\mathsf{Anon}_3 = 2$, then the adversary may be viewed as capable of tracing some element in round 3 back, roughly speaking, to a set consisting of two input elements. If $\mathsf{Anon}_k = n$, then the adversary has no information about the correspondence between inputs and elements in mixing round k, while $\mathsf{Anon}_k = 1$ indicates exact knowledge in round k about which input corresponds to a particular output. (The value $\log(\mathsf{Anon}_k)$ is a lower bound on the definition of anonymity given in [23].)

Our definition of anonymity needs to be adjusted a little bit for the last round. This is because we assume that the adversary learns some input/output correspondences; thus, in particular, $\mathsf{Anon}_R(s) = 1$ for any $s \in \mathcal{A}(O)$. We therefore define $\mathsf{Anon}_R = \min_{s \notin \mathcal{A}(O)} \mathsf{Anon}_R(s)$.

Clearly an input element enjoys no anonymity prior to mixing. This is stated formally in the following proposition:

Proposition 2.1. *We have $P_{\mathcal{A}}(0, s, s') = 1$ if $s = s'$ and $P_{\mathcal{A}}(0, s, s') = 0$ otherwise.*

The next proposition states that when an honest server takes a collection of input elements, mixes them, and then outputs them, it obscures the input/output relations on that group perfectly in the view of the adversary.

Proposition 2.2. *Assume that mix server j is not controlled by $\mathcal{A}$. In round k, mix server j processes $S_k(j)$. For all slot $s_1 \in S_k(j)$, we have*

$$P_{\mathcal{A}}(k+1, s_0, s_1) = \frac{1}{|S_k(j)|} \sum_{s \in S_k(j)} P_{\mathcal{A}}(k, s_0, s).$$

If, by contrast, a set of elements is mixed by a server controlled by $\mathcal{A}$, then the mixing operation does not decrease the ability of the adversary to trace any of these elements. Formally:

Proposition 2.3. *Assume that mix server j is controlled by $\mathcal{A}$. In round k, mix server j mixes the ciphertexts that occupy the slots in $S_k(j)$. For all $s_1 \in S_k(j)$, we have $P_{\mathcal{A}}(k+1, s_0, f(s_1)) = P_{\mathcal{A}}(k, s_0, s_1)$, where $f(s_1)$ is the slot in which lands the ciphertext that occupied slot s_1 before mix server j mixed $S_k(j)$.*

3. PARALLEL MIXNET PROTOCOL

Recall that $M \geq 2$ denotes the number of mix servers and n denotes the number of inputs. To simplify the presentation of our parallel mixnet protocol, we assume that $M^2 \mid n$. It is clear that this assumption can be relaxed without difficulty by adding dummy input elements to the mixnet.

Our proposed construction starts initially with an equal partitioning of slots among servers. We call a round in which slots are evenly partitioned among servers an equi-round.

Definition 3.1. **(Equi-round)** *Assume that M divides n. We say that mixing round k is an equi-round if $|S_k(j)| = n/M$ for all $j = 1, \ldots, M$.*

Our construction involves two basic operations. We refer to the first as a *rotation*. A rotation simply means that the subset of slots assigned to server j in round k is assigned to server $j+1$ in round $k+1$. In other words, each server passes its assigned slots to the next one in round-robin fashion. More formally:

Rotation: Mixing round $k+1$ is a rotation of mixing round k if $S_{k+1}(j) = S_k((j-1) \mod M)$ for all $j = 1, \ldots, M$.

The second basic operation in our construction is a *distribution*. In a distribution, the slots of every server are assigned uniformly across all servers in the next round.

Distribution: Assume that $M^2 | n$ and that mixing round k is an equi-round. We say that mixing round $k+1$ is a distribution of mixing round k if $|S_{k+1}(j) \cap S_k(j')| = n/M^2$ for all $j, j' \in \{1, \ldots, M\}$. Observe that this implies that round $k+1$ is also an equi-round.

Recall that in our model, an adversary $\mathcal{A}$ may control a number of mix servers and learn an arbitrary number of input/output relationships (up to $n-2$). Let $M' < M$ be the number of mix servers controlled by $\mathcal{A}$ (if $M' = 0$, we have an external adversary). Figure 1 gives our proposed parallel mixnet protocol.

Note first that the definition of Figure 1 does not specify a unique parallel mixnet protocol in terms of the subsets of slots $S_k(j)$ given to server j in round k. Rather, there is a family of mixnet protocols that satisfy that definition. We will show however that all the mixnets that satisfy our definition are equal up to isomorphism.

To simplify our discussion, we introduce first some new notation. Let $S = \{S_k(j)\}$ be a parallel mixnet defined by the four steps of Figure 1. For $j = 1, \ldots, M$, we let $B_S(j) = S_1(j)$ and $C_S(j) = S_{M'+2}(j)$. When no confusion is possible, we omit the subscript S and write simply $B(j)$ for $B_S(j)$ and $C(j)$ for $C_S(j)$.

Parallel mixnet construction

1. The first round of mixing is an equi-round. Slots are assigned to servers according to our ideal source of randomness (i.e. in practice randomness selected jointly by all the servers).
2. The next M' rounds of mixing are each a rotation of the previous round.
3. Round $M'+2$ is a distribution of round $M'+1$.
4. The next M' rounds of mixing are each a rotation of the previous round.

Figure 1: Our proposed parallel mixnet

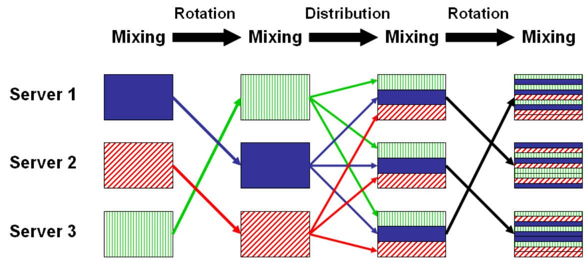

Figure 2: A parallel mixnet with 3 servers ($M = 3$) and $M' = 1$. The set $S_1(1)$ of slots originally assigned to Server 1 is colored solid blue, the set $S_1(2)$ is diagonally striped red and $S_1(3)$ is vertically striped green.

Intuition:

In step 1 of our parallel mixnet protocol, the slots are divided at random into "buckets" $B(1), \ldots, B(M)$ that are initially assigned to servers 1 through M. Since we assume an ideal source of randomness, the adversary has no control over the initial division of slots into buckets. Every server mixes the bucket is has been assigned.

In step 2, consisting of M' rounds, the servers pass their buckets of slots on to one another in round-robin fashion. The key observation is that by the end of step 2, every bucket will be processed by at least one honest server, since the total number of rounds of mixing in steps 1 and 2 is $M'+1 > M'$. As a consequence, therefore, of Proposition 2.2, the elements found in the slots of any given bucket after the end of step 2 will be completely mixed in the view of the adversary. This means that a given element is effectively traceable to an input set of size at least n/M. (In particular we have $\mathsf{Anon}_k \geq n/M$ after this step.)

To achieve mixing *among* buckets, we invoke a distribution operation in step 3. The original buckets are broken up and the slots are re-distributed into a new set of buckets $C(1), \ldots, C(M)$. After this step, every bucket $C(i)$ will contain an equal number of slots from each of the original set of buckets $\{B(i)\}$ created in step 1. This does not increase the level of anonymity for particular elements, but lays the groundwork for mixing to be completed in the next step.

In step 4, the new set of buckets $\{C(i)\}$ is passed in round-robin fashion exactly as in step 2. Again, each bucket will be processed by at least one honest server, since the total number of rounds of mixing in steps 3 and 4 is $M'+1 > M'$. Thanks to the distribution operation in step 3, the result will be a complete mixing of elements. In particular, since each bucket $C(i)$ includes an equal number of slots from each original bucket in $\{B(j)\}$, the adversary cannot trace a given output element to its original bucket $B(j)$. Since each original bucket $B(j)$ was mixed by at least one honest server, the adversary cannot trace a given output to a corresponding input with probability better than a random guess.

Implementation:

Let us illustrate with an example the choice of values for the various parameters of our scheme. The following discussion should also help clarify how our parallel re-encryption mixnet protocol compares with a sequential synchronous re-encryption mixnet in terms of performance.

Suppose that we want to mix the ballots cast in an election. The number of ballots (or inputs), n, is given to us. Both sequential mixnets and our parallel mixnet guarantee the privacy of ballots provided that the following condition holds: "Among the M mix servers involved in the mixing, at most M' are corrupt".

The choice of values for M and M' depends on a number of factors, such as the importance of the election, the reputation of servers based on past behavior, etc. All these factors are unrelated to the mixing protocol. Indeed, our confidence in the honesty of a server most likely does not depend on the type of protocol that the server is asked to run. The choice of the parameters M and M' is thus unaffected by whether we mix the ballots with a sequential mixnet or a parallel mixnet. For example, we may choose to use $M = 10$ servers and assume that no more than $M' = 6$ of those will collude together.

After choosing values for M and M', we may mix the ballots with a sequential synchronous mixnet, at a cost of Mn steps. We may also mix the ballots with a parallel mixnet protocol at a cost of at most $2n$ steps (the exact cost depends on the values M and M' and is given in figure 3). Our parallel mixnet is thus at least $M/2$ times more efficient than a sequential synchronous mixnet, yet it achieves very nearly the same privacy properties (see discussion below).

4. SECURITY PROOFS

We now prove that our mixnet offers optimal anonymity in the case where the adversary does not learn any input/output relations, i.e., when $|\mathcal{A}(I)| = 0$. We also show that even when the adversary does learn input/output relations, the mixnet still achieves near optimal anonymity given a large enough input set. To begin with, we show that our proposed construction is uniquely defined.

PROPOSITION 4.1. *The sequence of operations specified in Figure 1 defines a unique parallel mixnet up to isomorphism.*

PROOF. Let $\{S_k(j)\}$ and $\{T_k(j)\}$ be two sequential mixnets that satisfy the properties above. Let $\sigma_{j_0,j_1} = B_S(j_0) \cap C_S(j_1)$ and let $\tau_{j_0,j_1} = B_T(j_0) \cap C_T(j_1)$. The sets $\{\sigma\}$ and $\{\tau\}$ are both partitions of $\{1, \ldots, n\}$. Furthermore, since round $M'+2$ is a distribution of round $M'+1$ we have

	parallel mixnet	sequential mixnet
Total mixing time	$2n(M'+1)/M$	nM
Comp. cost per server	$2n(M'+1)/M$	n
Mixing rounds	$2(M'+1)$	M
Anonymity	$n-\lvert\mathcal{A}(I)\rvert$	$n-\lvert\mathcal{A}(I)\rvert$

Figure 3: Properties of parallel and sequential mixnets, for n inputs and M mix servers (of which up to M' may be under adversarial control)

$|\sigma_{j_0,j_1}| = |\tau_{j_0,j_1}| = n/M^2$ for all j_0, j_1. Therefore there exists a permutation π on n elements which maps σ_{j_0,j_1} to τ_{j_0,j_1} for all j_0, j_1. It is easy to see that π defines an isomorphism between $\{S_k(j)\}$ and $\{T_k(j)\}$. $\square$

Figure 3 summarizes the properties of our parallel mixnet protocol. The total mixing time and computational cost per server (expressed in number of steps) are simple to compute. The number of mixing rounds is the number of times that a message is exchanged between mix servers. We now prove the properties of our parallel mixnet with respect to anonymity.

Recall that $\mathcal{A}$ learns the set of slots $\mathcal{A}(I)$ that contain the input elements controlled by the adversary before the mixing begins, and the set of slots $\mathcal{A}(O)$ that contain the same elements after mixing is complete. For $j = 1, \ldots, M$, let $\alpha(j) = |\mathcal{A}(I) \cap B(j)|$ be the number of slots that are in input bucket $B(j)$ and contain initially an input element controlled by the adversary. Similarly, let $\gamma(j) = |\mathcal{A}(O) \cap C(j)|$. We also define $\delta(j_0, j_1)$ to be the number of slots that belong to input bucket $B(j_0)$ and to output bucket $C(j_1)$ and that contained an element controlled by the adversary at the end of step 2 of the parallel mixing protocol.

THEOREM 4.2. *Let $s_0 \in B(j_0)$ and $s_1 \in C(j_1)$. We have*

$$P_{\mathcal{A}}(R, s_0, s_1) = \frac{n/M^2 - \delta(j_0, j_1)}{\Big(n/M - \alpha(j_0)\Big)\Big(n/M - \gamma(j_1)\Big)}$$

PROOF. Let p_1 be the probability, in the view of $\mathcal{A}$, that the input element that originally occupied slot s_0 was moved to one of the slots that land in output bucket $C(j_1)$ during the distribution step. Let p_2 be the probability, in the view of $\mathcal{A}$, that an element that lands in bucket $C(j_1)$ after the distribution step ends up exactly in slot s_1 after the mixing is complete. By definition, we have $P_{\mathcal{A}}(R, s_0, s_1) = p_1 p_2$. Let us now compute p_1 and p_2.

The first M' rounds of mixing ensure that every input bucket is mixed at least once by an honest server. Thus at the time of the distribution, the content of every input bucket is uniformly distributed in the view of $\mathcal{A}$ across all n/M inputs, i.e, the adversary knows what is in the bucket, but not the ordering. In the distribution step, we transfer elements from input buckets to output buckets. Each output bucket contains an equal mixture of elements from each input bucket. Let us consider input bucket $B(j_0)$. The adversary $\mathcal{A}$ knows to which output bucket $\alpha(j_0)$ elements of $B(j_0)$ were assigned. The remaining $n/M - \alpha(j_0)$ are assigned uniformly at random in the view of $\mathcal{A}$. Since $\mathcal{A}$ knows that $\delta(j_0, j_1)$ elements from $B(j_0)$ were assigned to $C(j_1)$, we have $p_1 = (n/M^2 - \delta(j_0, j_1))/(n/M - \alpha(j_0))$.

The M' rounds of mixing that follow the distribution ensure that every output bucket is mixed at least once by an honest server. Thus all elements that are not in a slot that belongs to $\mathcal{A}(O)$ are uniformly distributed in the view of $\mathcal{A}$ and it follows immediately that $p_2 = 1/(n/M - \gamma(j_1))$. $\square$

Assuming that the adversary learns no input/output relation, we obtain optimal anonymity

COROLLARY 4.3. *If $\mathcal{A}(I) = \emptyset$, then our parallel mixnet has anonymity n with respect to $\mathcal{A}$.*

PROOF. If $\mathcal{A}(I) = \emptyset$, we have $\alpha(j_0) = \gamma(j_1) = \delta(j_0, j_1) = 0$ for all j_0, j_1. By Theorem 4.2, it follows that $P_{\mathcal{A}}(R, i, i') = 1/n$ for all i, i' and thus the mixnet has anonymity n. $\square$

We have shown that when the adversary learns no input/output relations, the mixnet offers optimal anonymity. What happens when the adversary does learn such relations, however? In this case, we can still achieve near optimal anonymity when the elements controlled by the adversary are uniformly distributed across buckets (or nearly so). We show this in the following corollary.

COROLLARY 4.4. *If the variables $\alpha(j_0), \gamma(j_1)$ and $\delta(j_0, j_1)$ are equal to their mean values for all j_0, j_1, our parallel mixnet has anonymity $n - |\mathcal{A}(I)|$.*

PROOF. We have $E(\alpha(j_0)) = E(\gamma(j_1)) = |\mathcal{A}(I)|/M$ and $E(\delta(j_0, j_1)) = |\mathcal{A}(I)|/M^2$. By Theorem 4.2, it follows that $P_{\mathcal{A}}(R, i, i') = 1/(n - |\mathcal{A}(I)|)$ for all i, i' and therefore the mixnet has anonymity $n - |\mathcal{A}(I)|$. $\square$

Statistically approximating ideal anonymity.
Thanks to our assumption of ideal randomness, the elements controlled by the adversary are statistically likely to be distributed in a roughly even fashion across buckets. In particular, the random variables $\alpha(j_0)$ and $\gamma(j_1)$ can be approximated for all j_0, j_1 by a Poisson distribution with mean $|\mathcal{A}(I)|/M$ and standard deviation $\sqrt{|\mathcal{A}(I)|/M}$. The random variable $\delta(j_0, j_1)$ can be approximated by a Poisson distribution with mean $|\mathcal{A}(I)|/M^2$ and standard deviation $\sqrt{|\mathcal{A}(I)|/M^2}$. The Chernoff bound shows that for a Poisson distribution V with mean $E(V)$ and standard deviation σ, $\Pr[|V - E(V)| > \sigma\epsilon] \leq 2e^{-\epsilon^2/4}$. It follows that with high probability the variables α, γ and δ are "close" to their means and therefore Corollary 4.4 is almost always a good approximation of the anonymity afforded by the mixnet.

5. PROOF OF OPTIMALITY

In this section, we now briefly prove that our construction achieves optimal efficiency. Our measure of efficiency is the total time required for the mixnet to complete the mixing and re-encryption operations (we do not consider the time required for the final, separate, decryption step that comes at the end). Other measures of efficiency, such as the amount of computation per server are possible, but we do not aim to optimize these. Our construction is optimal in this sense to within a factor of two, as we show in the following proposition.

PROPOSITION 5.1. *let $\mathcal{A}$ denote an adversary that controls $M' < M$ mix servers. A parallel mixnet with anonymity $\mathsf{Anon}_R > 1$ with respect to $\mathcal{A}$ must have total mixing time at least $n(M'+1)/M$ steps.*

PROOF. For every slot $s \in \{1, \ldots, n\}$, we define $f(s) = |\{j \in \{1, \ldots, M\} \mid \exists k \text{ s.t. } s \in S_k(j)\ \}|$. If there exists s such that $f(s) \leq M'$ then by Proposition 2.3, we know that there must exist s' such that $P_{\mathcal{A}}(R, s, s') = 1$ and therefore the anonymity of the mixnet with respect to $\mathcal{A}$ is $\mathsf{Anon}_R = 1$ contrary to our assumption. This proves that $f(s) \geq M' + 1$ for all s. Let $\mathsf{Comp}(j)$ denote the total number of steps that server j must execute. We have just proved that $\sum_{j=1}^{M} \mathsf{Comp}(j) \geq n(M' + 1)$. Since the total mixing time is greater than $(1/M) \sum_{j=1}^{M} \mathsf{Comp}(j)$, it follows that the total mixing time is at least $n(M' + 1)/M$. $\square$

In the special case where the adversary controls a maximal number of servers, we can show exact optimality.

PROPOSITION 5.2. *let $\mathcal{A}$ be an adversary that controls $M' = M-1$ mix servers. A parallel mixnet with anonymity $\mathsf{Anon}_R = n$ with respect to $\mathcal{A}$ must have total mixing time at least $2n$. Therefore our parallel mixnet protocol achieves optimal total mixing time when $M' = M - 1$.*

PROOF. Let $U = \{j \in \{1, \ldots, M\} \mid \exists k \text{ s.t. } |S_k(j)| = n\}$ be the set of servers that are given all n slots in at least one round of mixing. If server j is given all n slots in round k (i.e. $|S_k(j)| = n$), then all other servers must be idle in that round ($S_k(j') = \emptyset$ for all $j' \neq j$). The mixing time for the servers in U is thus $n|U|$.

Now let us consider a server j such that $j \notin U$. We claim that the computational cost $\mathsf{Comp}(j) = \sum_{k=1}^{R} |S_k(j)| \geq 2n$. The proof is by contradiction. Assume that $\mathsf{Comp}(j) < 2n$. Since there are n slots, there exists a slot s which is processed by server j in only one round k_0 (and recall that in that round $|S_{k_0}(j)| < n$ since $j \notin U$). Now if $\mathcal{A}$ controls all servers but j, we have $\mathsf{Anon}_R(s) < n$. Therefore the mixing time for servers not in U is at least $2n(M - |U|)/(M - |U|)$.

Altogether, the total mixing time is at least $n|U| + 2n(M - |U|)/(M - |U|)$. The total mixing time is minimized when $U = \emptyset$ and in that case it is exactly $2n$. $\square$

6. CONCLUSION

We have shown that re-encryption mix networks are not just highly distributed cryptographic constructions, but also highly parallelizable ones. Through distribution of small mix batches among servers and careful scheduling, we show how to reduce overall mixing time from $O(nM)$ to $O(n)$.

Our proposal engenders an important practical problem. While we expect that communications latency will be dwarfed by cryptographic processing time in many applications, e.g., elections, this premise requires empirical validation. For this reason, we would particularly like to see our construction implemented. Our proposal also engenders a theoretical problem. We are able to prove the optimality of our construction, in general, only within a factor of two. Whether a better proof or a better construction will lead to exact optimality is an open question.

7. REFERENCES

[1] M. Abe. Mix-networks on permutation networks. In *ASIACRYPT '99*, pages 258–273.

[2] M. Abe and F. Hoshino. Remarks on mix-network based on permutation networks. In *PKC '01*, pages 317–324.

[3] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography (extended abstract). In *PODC '00*, pages 123–132.

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

[5] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *IEEE Symposium on Security and Privacy '03*, pages 2–15.

[6] Y. Desmedt and K. Kurosawa. How to break a practical mix and design a new one. In *EUROCRYPT '00*, pages 557–572.

[7] R. Dingledine, V. Shmatikov, and P. Syverson. Synchronous batching: From cascades to free routes. In *Privacy Enhancing Technologies '04*. To appear.

[8] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In *Financial Cryptography '02*, pages 16–30.

[9] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *CRYPTO '01*, pages 368–387.

[10] P. Golle and D. Boneh. Almost entirely correct mixing with applications to voting. In *ACM CCS '02*, pages 68–77.

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

[12] M. Jakobsson. A practical mix. In *EUROCRYPT '98*, pages 448–461.

[13] M. Jakobsson and A. Juels. An optimally robust hybrid mix network. In *PODC '01*, pages 284–292.

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

[15] M. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX'02*, pages 339–353.

[16] A. Kiayias and M. Yung. The vector-ballot e-voting approach. In *Financial Cryptography '04*, pages 72–89.

[17] D. Knuth. *The Art of Computer Programming, Vol 3: Sorting and Searching*. Addison-Wesley, 1998.

[18] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1995.

[19] A. Neff. A verifiable secret shuffle and its application to e-voting. In *ACM CCS '01*, pages 116–125.

[20] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *ICICS '97*, pages 440–444.

[21] M. Ohkubo and M. Abe. A length-invariant hybrid mix. In *ASIACRYPT '00*, pages 178–191.

[22] C. Rackoff and D. R. Simon. Cryptographic defense against traffic analysis. In *ACM Symposium on Theory of Computing*, pages 672–681, 1993.

[23] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In *Privacy Enhancing Technologies '02*, pages 41–53.

[24] A. Waksman. A permutation network. *JACM*, 15(1):159–163, 1968.

[25] M. Wright, M. Adler, B. N. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. In *NDSS '02*, pages 39–50.