Millimix: Mixing in Small Batches

Markus Jakobsson[1] and Ari Juels[2]

[1] Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA
E-mail: `markusj@research.bell-labs.com`
[2] RSA Laboratories
RSA Security, Inc.
Bedford, MA 01730, USA
E-mail: `ajuels@rsasecurity.com`

Abstract. A *mix network* is a cryptographic construction that enables a group of players to permute and re-encrypt a sequence of ciphertexts so as to hide the relationship between input and output ciphertexts. In this paper, we propose a mix network known as *Millimix*. In constrast to other proposed constructions, Millimix enjoys a very high level of computational efficiency on small input batches, that is, batches of several thousand items or smaller. Additionally, Millmix possesses the full set of properties typically sought, but generally unavailable in many other mix network constructions, including public verifiability, robustness against malicious coalitions of players, and strong privacy guarantees. Millimix therefore promises to serve as a useful and practical complement to existing mix network constructions.

1 Introduction

A *mix network* is a cryptographic construction that enables one or more players to take a sequence of encrypted input messages, re-encrypt them, and output them in an unrevealed, randomly permuted order. The original conception of mix networks is due to Chaum [5], who envisioned them as a tool for privacy enhancement. They have since been proposed for use in such tasks as originator-anonymous e-mail [5], Web browsing [10], and secure elections [23, 25], as well as for seemingly unrelated applications such as anonmyous payment systems [19] and secure multiparty computation [18]. Mix networks may also serve implicitly in high level protocols as the basis of an anonymous channel. In many cases, such as anonymous Web browsing, they are perhaps the only realistic mechanism for achieving strong privacy in an untrustworthy environment.

In the past couple of years, researchers have focused their efforts on the investigation of *threshold* mix networks. These are mix networks implemented by multiple players, sometimes referred to as *mix servers*, in which correctness and privacy are assured even in the face of malicious server coalitions. Early efforts in this area include those of Ogata, Kurosawa, Sako, and Takatani [22], followed by Abe [1] and Jakobsson [15]. Early proposals for threshold mix networks were rather inefficient, requiring tens of exponentiations per item under a typical parameterization. Jakobsson [16] subsequently proposed a mix network that is about an order of magnitude faster than these previous ones, but only with the use of very large input batches – on the order of 100,000 items. A mix network more efficient on small batches is proposed by Abe in [2]. Developed independently of the construction presented here, that of Abe is very similar, but less computationally efficient.

Many applications require mixing of small batches. In the receipt-free election scheme of Hirt and Sako, for instance, n is equal to the number of electoral candidates [14]. For the secure multiparty computation technique proposed by Jakobsson and Juels, $n = 4$ would represent the input size required for the simulation of a two-input boolean gate [18]. Our goal in this paper is to propose a threshold mix network scheme that is highly efficient for input batches of this small size. We measure efficiency in practical terms. In particular, the asympotic complexity of our construction for an n-item mix is $O(n \log n)$, in contrast to $O(n)$ for other mix networks proposed in the literature. Our aim, however, is to drive the constant per-item costs for our mix to a level that is quite low by comparison, thus yielding a much more practical algorithm for small n. Our scheme has an additional advantage over constructions such as that in [16], namely the property of *public verifiability*. This is to say that players external to the mix network can easily verify the correct behavior of participating mix servers. We call our mix network scheme *Millimix*.

1.1 Background and overview of Millimix

As with any mix network, the aim of Millimix is to take as input a sequence of El Gamal encrypted ciphertexts $E = \{E_1, E_2, \ldots, E_n\}$ and output an El Gamal ciphertext sequence $E' = \{E'_1, E'_2, \ldots, E'_n\}$. The plaintexts corresponding to the sequence E' represent a random permutation of those corresponding to the sequence E. In other words, for some (secret) permutation σ selected uniformly at random, every ciphertext E'_i has the same associated plaintext as $E_{\sigma(i)}$. Briefly stated, then, E'

represents a random and secret permutation and re-encryption of the ciphertexts in E.

Millimix is based on an architecture known as a *comparison network*, which consists of a collection of *wires* and *comparitors*. A comparitor is a device that takes an ordered pair of integers (x, y) and outputs an ordered pair of integers $(x', y') = f(x, y)$. The *ordering* function f typically either swaps the input pair or else leaves it unchanged, according to some pre-specified condition. For example, in a comparison network that performs sorting, comparitors are constructed such that $f(x, y) = (x, y)$ if $x < y$ and $f(x, y) = (y, x)$ if $x \geq y$. A wire may transmit values into or out of the network or else between comparitors. A comparison network typically consists of n input wires, n output wires, and an arbitrary number of comparitors, with intermediate connecting wires. The network flow is unidirectional, i.e., no comparitor is used twice, and input values flow along wires in such a way that they are permuted in parallel by the network comparitors and then emerge on the output wires after a fixed number of comparisons. Note that different comparitors in the same network may employ different ordering functions f. Thus, it is convenient to let F denote the suite of potentially different re-ordering functions for all comparitors in the network, and to treat F as an input to the network, i.e, to regard comparitors as programmable.

Comparison networks may be constructed so as to achieve a variety of of different tasks. For our construction of Millimix, we employ what is known as a *permutation network*. The observation that a permutation network described in [28] is appropriate for a mix network is made in [2]. A permutation network takes inputs $x_1, x_2, \ldots, x_n$ and an easily calculable suite F_σ of re-ordering functions, where σ may represent any permutation on n items. Let $\sigma(i)$ denote the mapping of integer i under σ. Output from the mix network consists of the sequence of items $x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}$. Since there are $n!$ possibilities for σ, it is easy to see that a permutation network on n items must contain at least $\log_2 n! = \Omega(n \log n)$ comparitors. There is a simple general construction that contains exactly $n \log_2 n - n + 1$ comparitors. In this paper, we let Π denote this construction on n input items, where the value of n is left implicit. For further details on permutation and other types of comparitor networks, see [7, 28].

The basic idea behind Millimix is the following. Each mix server i in turn chooses a permutation σ_i uniformly at random and simulates the action of Π under F_{σ_i} on output ciphertexts generated by the previous mix server. (So as to avoid confusion with the notion of protocol simulation in proofs of zero-knowledge properties, we shall say that a mix server *exe-*

cutes Π or, analogously, *executes* a comparitor.) If all mix servers behave correctly, then the output ciphertext sequence E' for the mix network will represent the result of applying a permutation σ to the input ciphertext sequence E, where $\sigma = \sigma_k \circ \sigma_{k-1} \circ \ldots \circ \sigma_1$. Provided that the permutation σ_i of at least one participating mix server i can be made private, the permutation σ will remain so. This observation is the crux of the security properties of our construction.

Observe, however, that in the scheme as described here, comparitors change only the order, and not the values of input items as they pass through a comparitor. Thus any player can easily determine the permutation σ_i applied by mix server i. In order to obscure the permutations applied in Millimix, we make use of the semantically secure *re-encryption* property of El Gamal, as described in Section 3. For each comparitor it executes, each mix server re-encrypts the pair of output ciphertexts, thereby hiding the action of the comparitor. More formally, we may think of mix server i as implementing a suite of functions $F_{\sigma_i}^*$, where the associated reordering function for each comparitor not only reorders input values as in F_{σ_i}, but also randomly re-encrypts them.

A requirement on the full mix network is that the set of input plaintext values be identical to the set of output plaintext values. In fact, we maintain the set of plaintext values as an invariant throughout the execution of the network. In order to demonstrate that it has executed Π correctly, each mix server gives a proof, for each comparitor it executes, that the pair of input plaintext values is identical to the pair of output plaintext values. The main challenge addressed in this paper is to make this proof of correct comparitor execution efficient, as it represents the bulk of the computational work in Millimix. To do so, we elaborate on techniques first appearing in [17].

1.2   Organization

The remainder of this paper is organized as follows. In Section 2, we describe the security model underlying Millimix and discuss the features of our construction. In Section 3, we recall the details of some basic cryptographic primitives, and show how these are used to realize the building blocks underlying Millmix. We give details of the Millmix construction in Section 4. In Section 5, we present theorem statements on the security of our construction. We present some ideas for computational efficiency enhancement and efficiency analysis in Section 6.

2 Model and Goals

A mix network involves two types of participating entity: some number of *users* and *k mix servers*. We also assume the existence of a *bulletin board*. This is a publicly shared piece of memory to which all players have read access and appenditive, sequential write access with authentication.[1] Prior to invocation of the mix network, users post encrypted messages to the bulletin board. When some pre-determined triggering event occurs, e.g., a previously agreed upon number of messages has accumulated, the mix servers operate on the posted sequence of ciphertexts $E = \{E_1, E_2, \ldots, E_n\}$. They jointly permute and re-encrypt the originally posted ciphertexts, and write the resulting ciphertext sequence, $E' = \{E'_1, E'_2, \ldots, E'_n\}$, back to the bulletin board. For some permutation σ, ciphertext E_i shares the same plaintext as ciphertext $E'_{\sigma(i)}$; thus there is a one-to-one correspondance between input and output plaintexts. The mix servers may also post intermediate results to the bulletin board in the course of the mix computation to ensure public verifiability of correct behavior on the part of the mix servers.

There are a number of variants on this basic scenario. For example, the input messages to the bulletin board may be plaintexts, rather than ciphertexts. Alternatively, inputs may be ciphertexts and output messages, plaintexts. Overlap among players is also permissible: Users may participate as mix servers.

An *adversary* is a player who controls some number of the users and mix servers. Her aim is to compromise the private information of players and/or to corrupt the correct functioning of the mix network. As an example of the former, she might seek to compromise user privacy by finding a corresponding input/output message pair; as an example of the latter, she might seek to cause the mix network to output incorrect ciphertexts. An *active* cheater is a mix server whose output may be controlled entirely by the adversary. A *passive* cheater is one whom the adversary cannot cause to deviate from the protocol, but whose inputs and outputs the adversary may view. We define an *honest* mix server to be one that is in neither the active nor passive control of the adversary.

Millimix achieves the following security properties relative to a static adversary, that is, an adversary that fixes the set of players in its control prior to execution of the protocol. We assume that an adversary actively

---

[1] A bulletin board may, of course, be simulated or replaced by an authenticated broadcast channel or Byzantine agreement protocol [20]. In the latter case, the scheme is robust against an adversary actively corrupting fewer than one-third of the servers, rather than half.

corrupts fewer than $k/2$ of the mix servers and passively corrupts fewer than k. The adversary may control an arbitrary number of users. We denote as *negligible* any quantity that is asymptotically less than $1/poly$ for any polynomial *poly* in the system security parameters. We say that a player is *polynomial-time* if her computational resources are polynomially bounded in the system security parameters.

– **Public verifiability**: On reading the contents of the bulletin board at the end of the protocol, any polynomial-time player can identify any actively cheating mix server with probability negligably less than 1.

– **Privacy**: Millimix conceals the permutation σ. In particular, after any successful execution of Millimix, it is infeasible for an adversary to output a pair (I, O) of corresponding input and output ciphertexts with probability significantly greater than that yielded by a uniform, random guess. More precisely, for any i, the adversary cannot determine $\sigma(i)$ with probability non-negligibly greater than $1/n$. We prove privacy under the Decision Diffie-Hellman assumption.

– **Robustness**: It is infeasible for an adversary to cause Millimix to produce an incorrect output ciphertext sequence E' with non-negligible probability. We prove robustness under the discrete log assumption.

The novel property in Millimix is the following.

– **Computational efficiency on small batches**: The asymptotic computational cost per server is $\Theta(kn \log n)$. While this is greater than the $\Theta(kn)$ for previously proposed mix networks, the associated constants are very low, making this mix network substantially more efficient in practice for small n. Communication costs are also $\Theta(kn \log n)$, where posting to or reading from the bullentin board is assumed to incur one unit of communication cost.

3 Building blocks

3.1 Review of useful primitives

We construct Millimix around two cryptographic primitives, El Gamal encryption and the Schnorr identification protocol. We review these algorithms here.

El Gamal encryption Our first basic tool in Millimix is the El Gamal encryption algorithm [11], which works as follows. Let p be a large prime (typically 1024 bits long), and let $p = 2q + 1$ for another prime q. Let g be a generator of the unique subgroup of $\mathcal{G}_q$, the set of quadratic residues in Z_p^*. Given a secret key $x \in Z_q$, we define the corresponding public key to be the pair (y, g) where $y = g^x \bmod p$. (For the remainder of the paper, we shall implicitly assume computation in the multiplicative group Z_p where applicable.) To encrypt a message[2] $m \in \mathcal{G}_q$, we select an *encryption exponent* $\gamma \in_u Z_q$, where $\in_u$ denotes selection uniformly at random. The encryption consists of the pair $(\alpha, \beta) = (my^\gamma, g^\gamma)$. To decrypt using the secret key x, we compute $m = \alpha/\beta^x$.

The critical property that we make use of in the El Gamal cipher is that of *semantic security* (see, e.g., [21] for a formal definition). Intuitively, semantic security means that a ciphertext leaks no polynomial-time computable information about the corresponding plaintext. For our purposes, the most important consequence of this is that it is infeasible for an adversary to determine whether two ciphertexts encrypted under the same public key represent encryptions of the same plaintext. The ElGamal cipher is semantically secure under the Decision Diffie-Hellman assumption. See [27] for a discussion and proof of this property.

A player with knowledge of the El Gamal public key y, but not necessary of the corresponding private key x, can *re-encrypt* a ciphertext (α, β). Suppose that $(\alpha, \beta) = (my^{r_1}, g^{r_1})$ for some plaintext m and encryption exponent r. The player selects *re-encryption exponent* $r \in Z_q$ uniformly at random, and computes $(\alpha', \beta') = (my^{r_1} \times y^r, g^{r_1} \times g^r) = (my^{r_2}, g^{r_2})$, where $r_2 = r_1 + r$. The semantic security of El Gamal means that it is infeasible for another player to determine whether (α', β') and (α, β) represent the same plaintext.

Another useful property of the El Gamal encryption algorithm is that of *homomorphism*. If ciphertext (α_1, β_1) represents plaintext m_1 and (α_2, β_2) represents plaintext m_2, then the plaintext $m_1 m_2$ can be computed simply as $(\alpha_1\alpha_2, \beta_1\beta_2)$. Similarly $(\alpha_1/\alpha_2, \beta_1/\beta_2)$ represents an encryption of the plaintext m_1/m_2. Variants on Millimix can be constructed without the requirement for this property.

Note that El Gamal encryption may be straightforwardly performed over other group structures. Additionally, it is possible to use other ciphers with semantic security properties, such as the Goldwasser-Micali encryption scheme [13]. In certain mix network deployments, an addi-

---

[2] Messages not in $\mathcal{G}_q$ can be mapped onto $\mathcal{G}_q$ by appropriate forcing of the LeGendre symbol, e.g., inversion of the associated integer sign.

tional requirement, of which we omit discussion here, is that the cipher have a *non-malleable* variant. The El Gamal cryptosystem has such a variant. See [15] for further details.

Schnorr identification algorithm Our second basic tool is the Schnorr identification algorithm [26], which also operates over $\mathcal{G}_q$ as described above. The prover holds a private key $x \in Z_q$. The corresponding public key is (Y, G), where $y = G^x$, and $G \in \mathcal{G}_q$. To prove possession of the private key, the prover selects a value $e \in Z_q$ uniformly at random and sends a *commitment* $w = G^e$ to the verifier. The verifier responds with a random, l-bit challenge c. The prover sends as a response the value $s = xc + e$. The identification protocol can be converted into a signature algorithm, the Schnorr signature algorithm, by letting $c = h(w, m)$ for the message m to be signed. The prover verifies that $G^s = wy^c$. For further details, see, e.g., [21].[3] For l-bit challenges c, where $l = \Theta(\mathrm{poly}(|q|))$, the Schnorr identification protocol is an honest-verifier zero-knowledge proof of knowledge. For $l = \log(|q|)$, it is a zero-knowledge proof of knowledge, where $|q|$ denote the bit length of q.

3.2 Building blocks for Millimix

Most of the computational cost of Millimix derives from player proofs that comparitors have been correctly simulated. For this we use two building blocks, $\mathcal{PEP}$ and $\mathcal{DISPEP}$. The prototcol $\mathcal{PEP}$ enables a player to prove that an El Gamal ciphertext (α, β) a valid re-encryption of El Gamal ciphertext (α', β'). The protocol $\mathcal{DISPEP}$ enables a player to prove that one of two El Gamal ciphertexts (α_1, β_1) and (α_2, β_2) represents a valid re-encryption of an El Gamal ciphertext (α, β). Note that either of these protocols can be made interactive by replacing the use of Schnorr identification with a Schnorr signature, that is, by replacing challenges with hashes on prover commitments. We assume that all El Gamal encryptions take place with respect to a public key (y, g), where $y = g^x$ for private key x held distributively by participating mix servers.

Plaintext Equivalence Proof $\mathcal{PEP}$ Let us suppose that a player re-encrypts the El Gamal ciphertext (α, β) as (α', β'). In other words, for some plaintext m, the encryption $(\alpha, \beta) = (my^{\gamma_1}, g^{\gamma_1})$ and $(\alpha', \beta') = (my^{\gamma_2}, g^{\gamma_2})$ for some $\gamma_1, \gamma_2 \in Z_q$. The aim of $\mathcal{PEP}$ is for the player to use his knowledge

---

[3] What we describe here is in fact a generalization of the Schnorr identification protocol. Typically $G = g$ in standard implementations. Our generalization does not have any impact on the security of the algorithm.

of the re-encryption factor $\gamma = \gamma_2 - \gamma_1$ to prove that (α, β) and (α', β') represent the same plaintext.

We construct $\mathcal{PEP}$ by exploiting the homomorphism property of the El Gamal cipher. Observe, in particular, that if (α, β) and (α', β') represent the same plaintext, then $(\alpha/\alpha', \beta/\beta')$ represents an encryption of the plaintext value 1. Hence $\alpha/\alpha' = y^\gamma$ and $\beta/\beta' = g^\gamma$. We let $Y = (\alpha/\alpha')^z (\beta/\beta')$ and $G = y^z g$. Observe that $(\alpha/\alpha', \beta/\beta')$ may be regarded as a Schnorr public key (Y, G) whose corresponding private key is the re-encryption factor γ. The $\mathcal{PEP}$ algorithm is now implemented simply by having the prover perform the Schnorr identification algorithm on the public key (Y, G).

Given than $\mathcal{PEP}$ involves an execution of the Schnorr identification algorithm on publicly derivable values, it has the same zero-knowledge properties as the underlying algorithm. We therefore have the following lemma.

Lemma 1. *$\mathcal{PEP}$ is an honest-verifier zero-knowledge proof protocol.* $\square$

Let m_1 and m_2 be the plaintexts corresponding to (α, β) and (α', β') respectively. We have the following lemma, which states that the protocol is sound, i.e., successful cheating on the part of the prover in $\mathcal{PEP}$ is as hard as determining the secret value x. The claim is easily proven by demonstration of a knowledge extractor for x.

Lemma 2. *If $m_1 \neq m_2$, then the protocol $\mathcal{PEP}$ is a proof of knowledge of x.* $\square$

Other means of constructing $\mathcal{PEP}$ are possible. For example, it is possible to use the proof of equivalence of discrete logs that forms a component in the undeniable signature scheme of Chaum and [6]. The method we propose, however, is several times more computationally efficient.

Disjunctive Schnorr identification protocol The lynchpin of Millimix is what we refer to as a *disjunctive* Schnorr identification protocol. This is a variant on the Schnorr identification algorithm in which, rather than performing the protocol with respect to some public key (Y, G), the prover proves knowledge of the private key corresponding to at least one of two public keys, (Y_1, G_1) or (Y_2, G_2). The verifier, while capable of verifying the correctness of the protocol, is incapable of determining which private key the prover has knowledge of.

The disjunctive Schnorr identification algorithm works as follows. Let us assume w.l.o.g. that the prover knows the private key $x_1 = log_{G_1} Y_1$

associated with the key pair (Y_1, G_1). The prover chooses e_1 and s_2 at random and also a l-bit challenge c_2. He computes $w_1 = G_1^{e_1}$ and $w_2 = G_2^{s_2} Y_2^{c_2}$, and sends these values to the verifier. The verifier picks a random l-bit challenge c and sends it to the prover. The prover computes $c_1 = c \oplus c_2$ (where $\oplus$ denotes the bitwise XOR operation) and $s_1 = e_1 - c_1 x$, and sends s_1, s_2, c_1, and c_2 to the verifier. The verifier checks that $Y_i^{c_i} = G_i^{s_i} w_i$ for $i \in \{1, 2\}$. In essense, the prover can "cheat" on one of the two identification proofs exploiting the fact that it has one degree of freedom in its choice of challenges c_1 and c_2. This protocol enjoys the same ZK properties as a conventional Schnorr proof. It is easy to see that the proof is sound in the sense that the prover can only complete it successfully with knowledge of x_1 or x_2. This protocol may be made non-interactive or made into a signature algorithm by appropriate use of hash functions to replace the challenges. For further details, see [8, 9].

We do not make use of the disjunctive Schnorr identification algorithm directly in Millimix, but use it as a subroutine in the following building block.

Disjunctive plaintext equivalence proof $\mathcal{DISPEP}$ The protocol $\mathcal{DISPEP}$ enables a prover to demonstrate that an El Gamal ciphertext (α, β) represents a re-encryption of one of two different El Gamal ciphertexts, (α_1, β_1) or (α_2, β_2). We accomplish this by combining the protocol $\mathcal{PEP}$ with the disjunctive Schnorr identification algorithm in the obvious fashion. In particular, let $(Y_1, G_1) = (\alpha/\alpha_1, \beta/\beta_1)$ and $(Y_2, G_2) = (\alpha/\alpha_2, \beta/\beta_2)$. The protocol $\mathcal{DISPEP}$ simply involves the prover performing the disjunctive Schnorr identification (or signature) protocol with respect to the two public keys (Y_1, G_1) and (Y_2, G_2).

It is clear that $\mathcal{DISPEP}$ inherits the ZK properties of the underlying Schnorr identification protocol. We make use of following lemma.

Lemma 3. *$\mathcal{DISPEP}$ is an honest-verifier zero-knowledge proof protocol.*
$\square$

By demonstrating an appropriate knowledge extractor, we can easily prove the following. Let m denote the plaintext corresponding to ciphertext (α, β), and m_1 and m_2 denote the plaintexts corresponding respectively to (α_1, β_1) and (α_2, β_2).

Lemma 4. *Either $m = m_1$ or $m = m_2$, or the protocol $\mathcal{PEP}$ is a proof of knowledge of x.*
$\square$

4 Millimix Protocol: Details

4.1 Distributed key generation

The first step in the Millimix protocol is for the k servers to generate a joint El Gamal public key pair (y, g), where $y = g^x$. This public key is posted to the bulletin board. The private key x is then shared using (t, k)-threshold distributed key generation protocol, where $t = \lceil k/2 \rceil - 1$. Key generation may be achieved using the techniques described in the seminal paper of Pedersen [24] or in any one of a number of subsequent papers on threshold and proactive variants. See Gennaro *et al.* [12] for a brief overview of relevant distributed key generation techniques and some important caveats. Of course, determining x is no harder than computing a discrete log: an adversary need only compute $x = \log_g y$. In this paper, we shall make the following assumption about the distributed key generation protocol used in Millimix. This assumption applies to the techniques described in [24] and to may others of interest.

Assumption 1 *If a player can extract x from the transcript produced by the distributed key generation protocol, than she can solve discrete log problem.* □

4.2 Executing a comparitor

Let us now describe the protocol by which a mix server executes a comparitor. Input to the comparitor is a pair of El Gamal ciphertexts (α_1, β_1) and (α_2, β_2) on respective plaintexts m_1 and m_2. Output consists of El Gamal ciphertexts (α_1', β_1') and (α_2', β_2') corresponding to respective plaintexts m_1' and m_2'. After posting output to the bulletin board, the mix server must prove that either $(m_1, m_2) = (m_1', m_2')$ or else $(m_1, m_2) = (m_2', m_1')$. In other words, it must show that inputs were either swapped, or left in their original order, and then re-encrypted. To do this, the server proves the following two statements.

Statement 1: $m_1 = m_1'$ OR $m_1 = m_2'$.

Statement 2: $m_1 m_2 = m_1' m_2'$.

The mix server demonstrates the Statement 1 using direct application of $\mathcal{DISPEP}$. For Statement 2, he uses the homomorphic properties of

El Gamal to compute $E[m_1 m_2] = (\alpha_1 \alpha_1', \beta_1 \beta_1')$ and computes $E[m_1' m_2']$ analogously. He then invokes $\mathcal{PEP}$ as described in Section 3.[4]

4.3 Executing Millimix

Our description of Millimix is now simple. Mix server 1 takes the sequence of input ciphertexts $E = E^0 = E_1^0, E_2^0, \ldots, E_n^0$ and passes them through Π, executing each comparitor in turn as described in Section 1.1. He then proves the correctness of his exection of Π comparitor by comparitor. In the interactive version of the interactive version of the proof protocols, challenges are computed as the XOR of independent, randomly generated challenges contributed by each of the verifying mix servers. Proof transcripts are dynamically posted to the bulletin board. Mix server completes its execution of Π by writing to the bulletin board the output ciphertext sequence $E^1 = E_1^1, E_2^1, \ldots, E_n^1$, representing a random permutation and re-encryption of the ciphertexts in E^0. This process is repeated by every other mix server i in turn: mix server i takes input E^i and writes ciphertext sequence E^{i+1} to the bulletin board, along with the transcript of proofs of correctness. When mix server k posts $E^{k+1} = E'$ to the bulletin board, this completes the mix network operation.

All servers verify proofs of correct execution of Π as they are posted to the bulletin board by server i. If, at any time, a majority of players believe that server i has cheated, they expel him from the protocol, perform a resharing of the secret key x, and restart the protocol, with server $i + 1$ taking as input the the ciphertext sequence E^i. (If mix server k is determined to be cheating, then the last correctly formed ciphertext sequence is posted to the bulletin board as the output of the mix network.)

The observation that servers may execute complementary portions of Π and thereby reduce the total number of instantiations of Π is explored in [2].

5 Security

We now analyze the security of the interactive version of our contruction against an adversary that corrupts fewer than $k/2$ mix servers in an active sense and fewer than k in a passive sense. As described before, Schnorr protocol challenges are generated as the XOR of random strings generated

---

[4] It is possible to prove the correctness of a comparitor more straightforwardly by means, e.g., of a statement of the form $[(m_1 = m_1')$ AND $(m_2 = m_2')]$ OR $[(m_1 = m_2')$ AND $(m_2 = m_1')]$. Such approaches are rather less efficient, though.

by each verifying mix server. Thus, given at least one verifying server that has not been actively corrupted, challenges will be distributed uniformly at random. This means that in the described attack scenario, challenges will be distributed uniformly at random, as they would be if generated by a single honest verifier. Thus it suffices to prove our protocols zero knowledge under the assumption that the verifier is honest.

Lemma 5. *The distribution of any proof transcript for Π is simulatable by any verifying mix server.*

Proof (sketch): Since fewer than half of the players have been actively corrupted, no passively corrupted or honest mix server will be removed from participation in the protocol. Thus, at any time during the execution of the protocol, there will be at least one passively corrupted or honest verifying server.

Recall from Lemmas 1 and 3 that the proof protocols employed with the execution of Π are honest-verifier zero-knowledge. Although it does not immediately follow that the composition of these proofs is zero-knowledge, the entire set of proofs may in fact be simulated using the same techniques for independent simulation of $\mathcal{PEP}$ and $\mathcal{DISPEP}$. The lemma follows.   □

Theorem 1. *Millimix achieves privacy under the Decision Diffie-Hellman assumption.*

Proof (sketch): Semantic security means that there is no algorithm A of the following form. Algorithm A is given as input El Gamal encryptions of plaintexts m_0 and m_1, and a pair of random re-encryptions c_0 and c_1. With probability non-negligibly greater than $1/2$, algorithm A guesses b such that m_0 is the plaintext for c_b.

Since the adversary corrupts fewer than $k/2$ servers in an active fashion, at least one honest server will adhere correctly to and not be removed from participation in the mix network. Suppose that Millimix does not achieve the privacy assumption. Then for some set of input ciphertexts on plaintexts $m_0, m_1, \ldots, m_{n-1}$, the adversary is able to output a pair (I, O) of corresponding input and output ciphertexts with a non-negligible advantage over a random guess. It is easy to see, therefore, that with non-negligible probability, the adversary is able to determine the output ciphertext C corresponding to some fixed input item m_i with probability non-negligibly greater than $1/n$. It is also easy to see that there is an input item m_j such that the adversary outputs (m_j, C) with probability at most $1/n$. Assume, without loss of generality, that $i = 0$ and $j = 1$.

We now construct algorithm A as follows. Algorithm A encrypts m_0 and m_1 and simulates their passage through Millimix. In place of the corresponding output ciphertexts for these two input items, A inserts random re-encryptions of c_0 and c_1. Algorithm A then invokes the adversary to guess at an associated input/output pair (I, O). If $I = m_0$ or $I = m_1$ and $O = c_b$ for $b \in \{0, 1\}$, then A outputs b. It is easy to see that b will be correct with probability non-negligibly greater than $1/2$. Since, by Lemma 5, all proofs in the protocol are zero knowledge, Algorithm A has computed b with no information other than that revealed by the El Gamal ciphertexts. This violates the semantic security assumption on the El Gamal cipher, and therefore the Decision Diffie-Hellman assumption. $\square$

Theorem 2. *Millimix is robust under the discrete log assumption.*

Proof (sketch): By Lemmas 2 and 4, it is infeasible for a server to cheat in its execution of Π without knowledge of the secret key x. Now, by Lemma 5, proof transcripts for Π are simulatable. Therefore, the robustness of our construction relies on the difficulty for the adversary of determining the secret key x given information derived from the distributed key generation protocol. By Assumption 1, it now follows that the hardness of determining x is equivalent to the hardness to the discrete log problem. $\square$

Given Theorem 2, combined with the fact that all transcripts are posted to the bulletin board, it is easy to see that our construction is publicly verifiable. Thus we state the following theorem without proof.

Theorem 3. *Millimix is publicly verifiable.* $\square$

6   Efficiency

6.1   Batch verification and encryption

Recall that every mix server i in Millimix must in turn execute Π. All other mix servers verify the correctness of the proofs generated by this mix server. In what follows, we refer to as the *prover* the mix server executing Π at a given time. We refer to any of the other mix servers as a *verifier*.

Verifiers in Millimix must check the correctness of many Schnorr identification transcripts – several for each comparitor in Π. This suggests

that the work of the verifiers might be reduced by employing batch verification techniques, methods proposed in the literature for accelerating signature verification by grouping underlying mathematical operations together (see, e.g., [3]). We demonstrate a method here for simultaneous batch verification of a collection of $\mathcal{PEP}$ and $\mathcal{DISPEP}$ proofs. We rely upon the use of addition chains to accelerate the verification process. Batch verification is of course not necessary for the security or robustness properties of Millimix, but improves its efficiency considerably.

Recall that the prover supplies verifiers with a set $\{w_i, c_i, s_i\}_{i=1}^{z}$ of purported Schnorr proofs on public keys $\{Y_i, G_i\}_{i=1}^{z}$, for some z. The verifier checks the correctness of a given proof by verifying that $G_i^{s_i} = Y_i^{c_i} w_i$. It is common practice to reduce the computation required for this equation by rewriting it as $G_i^{s_i} Y_i^{-c_i} = w_i$, and using simultaneous multiple exponentiation to compute the double exponentiation $G_i^{s_i} Y_i^{-c_i}$. This reduces the work of the verifier to the equivalent of about 1.17 exponentiations, rather than 2.

We may carry this idea a step further by grouping together a set of z Schnorr proofs and having the verifier check that:

$$\prod_{i=1}^{z} G_i^{s_i} Y_i^{-c_i} = \prod_{i=1}^{z} w_i. \tag{1}$$

Note that Eqn. 1 does not immediately imply the correctness of the set of individual equations $\{G_i^{s_i} = Y_i^{c_i} w_i\}_{i=1}^{z}$. For this, we rely on the following theorem. Let us denote by $\mathcal{BV}$ the interactive protocol in which the prover and verifier engage in a series of a $\mathcal{PEP}$ and b $\mathcal{DISPEP}$ proofs, and the verifier checks these proofs by means of Eqn. 1. Let Y_i, G_i be the key pair for $\mathcal{PEP}$ proof i, with secret key x_i and let $[Y_i^0, G_i^0, Y_i^1, G_i^1]$ be the key pair for $\mathcal{DISPEP}$ proof j, with secret keys x_j^0 and x_j^1. Proof of the following theorem relies on elementary construction of a knowledge extractor, and is therefore omitted.

Theorem 4. *The protocol $\mathcal{BV}$ is an honest-verifier zero-knowledge proof of knowledge of either x or else of $\{x_i\}_{i=1}^{a}$ and one of $\{x_j^0, x_j^1\}$ for all $j \in [1, \ldots, b]$.*

As an efficiency enhancement to batch verification, we can have verifiers use addition chains to check Eqn. 1 rapidly. Perhaps the best addition chain method for our purposes is that proposed by Bleichenbacher [4]. Of course, addition chain methods may also be used equally well to accelerate the work of the prover.

6.2 Efficiency analysis

For each comparitor it executes, a mix server must compute two El Gamal re-encryptions. This requires computation of the values g^{r_1}, g^{r_2}, y^{r_1}, y^{r_2}, where r_1 and r_2 are random re-encryption factors. Additionally, the mix server must perform one invocation of $\mathcal{PEP}$, and one of $\mathcal{DISPEP}$. For each $\mathcal{PEP}$ proof i, the prover must compute $G_i^{e_i}$ for a random value e_i. Since $G_i = y^{u_i} g^{v_i}$ for known values u_i and v_i, this involves a double exponentiation to fixed bases y and g. For each $\mathcal{DISPEP}$ proof, it is easy to see that the prover must compute two such double exponentiations. Thus, for each comparitor, the prover must compute three double exponentiations to fixed bases and four exponentiations to fixed bases. A verifying mix server must verify the transcripts yielded by one invocation of $\mathcal{PEP}$ and one of $\mathcal{DISPEP}$ – hence three Schnorr identification proofs – for each comparitor.

We shall consider a straightforward application of the techniques of [4] to improve the efficiency of both prover and verifier. In a mix network architecture with m comparitors, we shall assume the prover must perform $j = 7m$ multiple independent exponentiations. (That is, for the sake of simplicity, we shall treat double exponentiations as two separate exponentiations.) For this task, the number of modular multiplications t required by the techniques described in [4] is bounded above as follows:

$$t \leq \log_2 N + \frac{(j-1) \ln N}{\ln(j-1) - \ln\ln(j-1)},$$

where $N = 2^{|q|}$. The corresponding task of a verifying mix server is to perform simultaneous multiple exponentiation on $j = 3m$ terms. We have the following bound on number of modular multiplications t required for this task: $t \leq j - 1 + \log_2 N + \frac{(j-1) \ln N}{\ln(j-1) - \ln\ln(j-1)}$.

To provide a concrete sense of the efficiency of Millimix in a practical setting, let us consider some typical parameter settings. The El Gamal cipher is most commonly parameterized such that p is a 1024-bit prime and q is a 1023-bit prime. Thus full exponentiation of an element of $\mathcal{G}_q$ involves an average of 1534.5 modular multiplications. Table 1 below presents the total computational work for the prover and for each verifier. We measure computational effort in terms of the equivalent number of full exponentiations in $\mathcal{G}_q$, i.e., we divide the number of required modular multiplications by 1534.5. Numbers in parentheses indicate the work per input item. We denote the number of input items to the network by n, as above. (While we consider only values n that are powers of two;

other values of n are possible.) The heading d denotes the number of comparitors in the permutation network on n items.

n	d	Prover work (PW)	Verifier work (VW)
4	5	7.6 (1.9)	4.6 (1.1)
16	49	39.5 (2.5)	20.7 (1.3)
64	321	183.6 (2.9)	91.3 (1.4)
256	1793	806.9 (3.2)	390.1 (1.5)
1024	9217	3439.4 (3.4)	1635.6 (1.6)
4096	45057	14398.0 (3.5)	6772.8 (1.7)

Table 1. Computational work for Millimix

As each mix server must execute Π in turn, the total computational cost per mix server for Millimix is $PW + (k - 1)VW$. Hence, for example, with $k = 5$ mix servers and $n = 64$, the total computational cost per mix server is equivalent to about 122.3 full exponentiations, or about 7.6 exponentiations per input item.

It is undoubtedly possible to achieve better efficiency by exploiting the fact that the prover performs a majority of double exponentiations and also exponentiations with respect to fixed-base exponents. Additionally, it is possible to achieve large batch sizes, and thus better efficiency, by employing an "optimistic" approach to verification when $k \geq 3$. This is to say that a mix server batches together verification of all proofs performed by all other mix servers. In the presumably rare event that cheating is detected, the cheater is identified using a more expensive verification procedure. Another possible approach to making Millimix more efficient would be to depart from the hardware-based foundations of permutation networks and consider comparitors with more than two inputs. This might be an especially fruitful line of investigation for small mix networks, and mix networks where n is not a power of 2.

Acknowledgements

References

1. M. Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In K. Nyberg, editor, *EUROCRYPT '98*, pages 437–447. Springer-Verlag, 1998. LNCS No. 1403.

2. M. Abe. A mix-network on permuation networks, 1999. To appear in Asiacrypt '99.

3. M. Bellare, J.A. Garay, and T.Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *EUROCRYPT '98*. Springer-Verlag, 1998. LNCS No. 1403.

4. D. Bleichenbacher. Addition chains for large sets, 1999. Manuscript.

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

6. D. Chaum and H. van Antwerpen. Undeniable signatures. In G. Brassard, editor, *CRYPTO '89*, pages 212–216. Springer-Verlag, 1989. LNCS No. 435.

7. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. Mc-Graw Hill, 1994.

8. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO '94*, pages 174–187. Springer-Verlag, 1994. LNCS No. 839.

9. A. de Santis, G. di Crescenzo, G. Persiano, and M. Yung. On monotone formula closure of SZK. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 454–465. IEEE Press, 1994.

10. E. Gabber, P. Gibbons, Y. Matias, and A. Mayer. How to make personalized Web browsing simple, secure, and anonymous. In R. Hirschfeld, editor, *Financial Cryptography '97*, pages 17–31, 1997.

11. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.

12. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *Proc. EUROCRYPT '99*, pages 295–310, 1999.

13. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comp. Sys. Sci*, 28(1):270–299, 1984.

14. M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption, 1999. Manuscript.

15. M. Jakobsson. A practical mix. In K. Nyberg, editor, *EUROCRYPT '98*, pages 448–461. Springer-Verlag, 1998. LNCS No. 1403.

16. M. Jakobsson. Flash mixing. In *Proc. of 1999 ACM Symposium on Principles of Distributed Computing (PODC)*, 1999. To appear.

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

18. M. Jakobsson and A. Juels. Mix and match: Secure multiparty computation using mix networks, 1999. Manuscript.

19. M. Jakobsson and D. M'Raïhi. Mix-based electronic payments. In *Selected Areas in Cryptography (SAC) '98*, 1998. To appear.

20. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1995.

21. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

22. W. Ogata, K. Kurosawa, K. Sako, and K. Takatani. Fault tolerant anonymous channel. In *Proc. ICICS '97*, pages 440–444, 1997. LNCS No. 1334.

23. C. Park, K. Itoh, and K. Kurosawa. All/nothing election scheme and anonymous channel. In T. Helleseth, editor, *EUROCRYPT '93*. Springer-Verlag, 1993. LNCS No. 921.

24. T. Pedersen. A threshold cryptosystem without a trusted third party. In D.W. Davies, editor, *EUROCRYPT '91*, pages 522–526. Springer-Verlag, 1991. LNCS No. 547.

25. K. Sako and J. Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In L.C. Guillou and J.-J. Quisquater, editors, *EUROCRYPT '95*. Springer-Verlag, 1995. LNCS No. 921.

26. C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

27. Y. Tsiounis and M. Yung. On the security of ElGamal-based encryption. In *1998 International Workshop on Practice and Theory in Public Key Cryptography (PKC '98)*, 1998. To appear.

28. A. Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968.