DIMACS Technical Report 99-33
June 1999

Millimix: Mixing in Small Batches

by

Markus Jakobsson[1]
Bell Laboratories
Murray Hill, NJ 07974

Ari Juels
RSA Laboratories
Bedford, MA 01730

[1]Permanent Member

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, Telcordia Technologies (formerly Bellcore) and NEC Research Institute.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We present *Millimix*, a mix network that is highly efficient on small input batches. The construction is conceptually simple, and both robust and private in the face of collusion by any minority set of malicious players. Additionally, Millimix the feature of being publicly verifiable. In other words, the mixing operation yields a transcript that demonstrates correctness to a third party that the mix proceeded correctly.

Keywords: El Gamal, mix network

1 Introduction

A *mix network* is cryptographic primitive that takes a collection of n encrypted input messages, re-encrypts them, and outputs them in an unrevealed, randomly permuted order. The original conception of mix networks is due to Chaum, who envisioned them as a tool for privacy enhancement. They have since been proposed for use in such tasks as originator-anonymous e-mail, Web browsing, and secure elections, as well as for seemingly unrelated applications such as anonmyous payment systems [9] and secure multiparty computation. 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 privacy in the presence of untrusted entities.

In the past couple of years, researchers have proposed a number of *threshold* mix networks. These are mix networks implemented by multiple *mix servers*, where the correctness and privacy of the mix is robust against malicious coalitions of servers. Early efforts included those of Ogata, Kurosawa, Sako, and Takatani [11], followed by Abe [1] and Jakobsson [6]. These mix networks were rather inefficient. Counting the number of exponentiations per item under a typical parameterization, their costs range between around fifty [6] and up to more than a thousand [11, 1]. Jakobsson [7] subsequently proposed a mix network bringing the cost down close to that of performing one exponentiation per input item. The two more efficient of these mix networks, however, while achieving a higher per-item efficiency than the others, assume very large input batches (of say, 1,000,000 items).

Many applications require mixing of small batches. In the receipt-free election scheme of Sako *et al.* [12], for instance, n is equal to the number of electoral candidates. For the secure multiparty computation techniques of Jakobsson and Juels [8], $n = 4$ would represent a typical input size. 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 $O(n \log^2 n)$, in contrast to $O(n)$ for other mixes. The constant per-item costs for our mix, however, are very low by comparison, and thus yield a much more practical algorithm for small n. We call our mix network scheme *Millimix*.

1.1 Sketch of Millimix

The aim of Millimix is to take a sequence of El Gamal encrypted ciphertexts $E = \{E_1, E_2, \ldots, E_n\}$ and output an El Gamal ciphertext sequence $E' = \{E'_1, E'_2, \ldots, E'_n\}$. For some (secret) permutation σ generated uniformly at random, the ciphertext E'_i has the same associated plaintext as $E_{\sigma(i)}$, but has been re-encrypted. In other words, E' represents a random and secret permutation and re-encryption of the ciphertexts in E.

Millimix is based on a construction known as a *sorting network*, which is typically viewed as 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 $(x', y') = f(x, y)$. Typically, f reorders integers based on their comparitive values. In other words $f(x, y) = (x, y)$ if $x < y$

and $f(x, y) = (y, x)$ if $x \geq y$. A sorting network consists of n input wires and n output wires, with intermediate wires to connect comparitors. 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. Comparitors are arranged in such a way that the sorting network sorts inputs in numerical order. In other words, it takes inputs $x_1, x_2, \ldots, x_n$ and outputs $x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}$ where σ is a permutation and $x_{\sigma(i)} \leq x_{\sigma(j)}$ for $i < j$. Since there are $n!$ possibilities for σ, it is easy to see that sorting network must contain at least $\log_2 n! = \Omega(n \log n)$ comparitors. General constructions are known that contain exactly $\frac{n}{4}(\log^2 n + \log n)$ comparitors. Constructions with fewer comparitors are known for specific small n. For further details on sorting and other types of relevant networks, see [2].

In Millimix, k mix servers may be viewed as simulating a sorting network S on n input items. In particular, mix server i takes as input a sequence of ciphertexts $E[m_1], E[m_2], \ldots E[m_n]$, where $E[m]$ denotes El Gamal encryption (under some encryption exponent) of plaintext m. The server generates a permutation σ_i uniformly at random. She then treats the ciphertexts as though passing them through the wires and comparitors of S. In the sorting network S, though, the comparitors "sort" items according to the permutation σ_i. In other words, if input to a given comparitor is the (possibly re-encrypted) ciphertext pair $(E[m_u], E[m_v])$, then the comparitor outputs $(E[m_u], E[m_v])$ if $\sigma^{-1}(u) > \sigma^{-1}(v)$; otherwise, it outputs $(E[m_u], E[m_v])$. The comparitor also re-encrypts items prior to outputting them, so that output items cannot be matched with input items. For each comparitor that mix server i simulates, she proves to all other mix servers that the output items of the comparitor are the same as the input items.

Mix server 1 performs this simulation of S on the ciphertext input $E = E_1, E_2, \ldots, E_n$, i.e., the input to the mix network, and passes her output to mix server 2. Each mix server i in turn takes its input from the previous one and repeats the process of simulating S using a random permutation σ_i. Mix server k outputs the ciphertext sequence $E' = \{E'_1, E'_2, \ldots, E'_n\}$. The permutation σ applied by the mix network as a whole is equal to the composition of the individual (and secret) permutations applied by the individual mix servers. In other words, $\sigma = \sigma_n \circ \sigma_{n-1} \circ \ldots \circ \sigma_1$. Since players prove that comparitors are correctly simulated, the mix network is robust against cheating. The fact that comparitors re-encrypt input items ensures privacy. Millimix can consequently achieve correct, private computation in the presence of a static, malicious adversary that corrupts fewer than half of the mix servers.

1.2 Organization

The remainder of this paper is organized as follows. In Section 2, we describe the security model in which we construct 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 cryptographic building blocks underlying Millmix. We give details of the Millmix construction in Section 4. We defer discussion of efficiency and proofs for the full version of this paper.

2 Model and Goals

A mix network involves two types of participating entity: *users* and *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.[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 possible 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. In particular, users may also participate as mix servers.

An *adversary* is a player who controls some number of the users and mix servers. Her aim is to compromise players over whom she does not exercize control, or to corrupt the correct functioning of the mix network. She may seek to compromise user privacy by linking an output message with an input message, or else cause an incorrect computation, e.g., one in which incorrect ciphertexts are output.

Millimix, the mix network we present in this paper, achieves the following security properties relative to a static adversary that corrupts fewer than half of the mix servers:

- **Public verifiability**: Any player (even a non-participatory one) can detect incorrect behavior on the part of any mix server with overwhelming probability using information posted to the bulletin board.
- **Robustness**: Millmix will produce correct output, i.e., a permutation (and re-encryption) of the input ciphertexts.
- **Privacy**: The mix network conceals the permutation σ. That is, it is infeasible for the adversary to determine whether a given output message corresponds to a given input message with probability significantly greater than that yielded by a uniform, random guess. More precisely, for any $\sigma(i)$, the adversary cannot determine i with probability $1/m + 1/poly(k)$, where $poly(k)$ denotes a polynomial in security parameter k.

The novel property in Millimix is the following.

[1] A bulletin board may, of course, be simulated or replaced by a broadcast channel. In fact, the scheme we present in this paper is secure under the rather more modest assumption of the availability of secure channels. We consider the bulletin board setting for simplicity of presentation.

- **Computational efficiency on small batches**: The asymptotic computational cost for the network given a fixed number of mix servers is $O(n \log^2 n)$. While this is greater than 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 $O(n \log^2 n)$.

3 Building blocks

3.1 Review of useful primitives

Millimix uses two basic 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 [5], which works as follows. Let q be a large prime (typically 160 bits long), and let $p = lq + 1$ be another large prime (typically 1024 bits long.) Let g be a generator of a subgroup of G_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 value m, 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., [10] 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 [13] for a proof of this property.

The property of *semantic security* enables a player with knowledge of the El Gamal public key y, but not necessary of the corresponding private key x, to *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 $(\alpha', \beta') = (my^{r_1} \times r^k, g^{r_1} \times g^r) = (my^{r_2}, g^{r_2})$, 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 $(\alpha_1/\alpha_2, \beta_1/\beta_2)$ is an encryption of the plaintext m_1/m_2.

Schnorr identification algorithm Our second basic tool is the Schnorr identification algorithm, which also operates over G_p 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 $G = g^k$, for some $k \in Z_q$. To prove possession of the private key, the prover selects a value $e \in Z_q$ uniformly at random and sends a *witness* $w = G^e$ to the verifier. The verifier responds with a challenge c, selected uniformly at random from Z_q. 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 futher details, see, e.g., [10].[2]

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 proof protocols, $\mathcal{DISPEP}$ and $\mathcal{PEP}$. The prototcol $\mathcal{PEP}$ enables a player to prove that an El Gamal ciphertext (α, β) represents a re-encryption of El Gamal ciphertext (α', β'). The protocol $\mathcal{DISPEP}$ enables a player to prover that one of two El Gamal ciphertexts (α_1, β_1) and (α_2, β_2) represents a re-encryption of an El Gamal ciphertext (α, β).

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 $(\alpha, \beta) = (my^{k_1}, g^{k_1})$ and $\alpha', \beta' = (my^{k_2}, g^{k_2})$ for some $k_1, k_2 \in Z_q$. The aim of $\mathcal{PEP}$ is for the player to use his knowledge of the re-encryption factor $k = k_2 - k_1$ to prove that (α, β) and (α', β') represent the same plaintext.

We construct $\mathcal{PEP}$ using 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 $\alpha/\alpha' = y^k$ and $\beta/\beta' = g^k$. We let $Y = (\alpha/\alpha')^{z_1}(\beta/\beta')^{z_2}$ and $G = y^{z_2}g^{z_1}$. 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 k. The $\mathcal{PEP}$ algorithm is now implemented simply by having the prover perform the Schnorr identification algorithm on the public key (Y, G). Note that the protocol can be made interactive by having the prover generate z_1 and z_2 using a hash function or pseudo-random function in an appropriate fashion and using the Schnorr signature algorithm instead of the identification algorithm.

The $\mathcal{PEP}$ algorithm is secure in the sense that it is infeasible for the prover to convince the verifier falsely of the equivalence of two plaintexts, and also in that it does not reveal significant information about the re-encryption factor k. Formal theorem statements and proof sketches are deferred for the full version of this paper.

[2] 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.

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 uses one of two public keys, either (Y_1, G_1) *or* (Y_2, G_2). The verifier, however, while capable of verifying the correctness of the protocol, is incapable of determining any information about with respect to which of these two keys the prover executed the protocol.

We first describe an interactive variant of this protocol, namely a *disjunctive Schnorr identification protocol*. Then we describe a non-interactive variant, namely a *disjunctive Schnorr signature protocol*.

The disjunctive Schnorr identification algorithm works as follows. Let us assume w.l.o.g. that the prover knows the secret $x_1 = log_{G_1} Y_1$ associated with the key pair (Y_1, G_1). Further, let us assume that all operations take place in a large group G of known order m. The prover chooses e_1 and s_2 at random and also a k-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 k-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 $i = 1, 2$. 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 zero knowledge properties as a conventional Schnorr proof. In particular, it is zero knowledge if $k = O(\log \log m)$, and, with use of commitments to enforce joint selection of a random challenge c can be made computational zero knowledge for $k = O(poly(\log m))$. 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 [3, 4].

We do not make use of the disjunctive Schnorr identification or signature algorithms directly in Millimix, but use one of them 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 a disjunctive Schnorr identification (or signature) algorithm with respect to the two public keys (Y_1, G_1) and (Y_2, G_2).

We shall make use of a final building block used by the verifier to check the multiple Schnorr proofs presented by the prover. We emphasize that this tool is not necessary for the security or robustness properties of Millimix, but improves its efficiency considerably.

Batch Schnorr verification Suppose a verifier is presented with a set $\{w_i, c_i, s_i\}_{i=1}^k$ of purported non-interactive Schnorr proofs on public keys $\{Y_i, G_i\}_{i=1}^k$. The verifier can verify each proof individually by checking that $G_i^{s_i} = Y_i^{c_i} w_i$. To perform batch Schnorr verification, the verifier checks that $\prod_i G_i^{s_i} = \prod_i Y_i^{c_i} \prod_i w_i$. Verification of this last equality may be accelerated considerably using the technique of *simulataneous multiple exponentiation* on the equality $\prod_i G_i^{s_i} \prod_i Y_i^{-c_i} = \prod_i w_i$. We show in the appendix that if it is infeasible for the prover to complete the batch Schnorr verification protocol successfully without knowledge of all private keys corresponding to the public keys.

4 Millimix

We are now ready to describe in detail the construction of Millimix.

4.1 Key sharing

The first step in the Millimix protocol is for the players to generate a joint El Gamal public key pair (y, g), where $y = g^x$. The private key x is shared additively. In other words, each mix server i holds share x_i and $x = \sum_i x_i$. Additionally, each mix server performs secret sharing on its share x_i using $(n, \lfloor n/2 \rfloor)$-Feldman VSS. (This must be done carefully, as in Canetti et al.) For further details, the reader is referred to the extensive literature on secret sharing protocols.

4.2 Simulating a comparitor

Let us now describe the protocol by which a mix server simulates 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 $(m_1, m_2) = (m_2', m_1')$. To do this, it suffices for the mix server to prove two equalities.[3] These equalities, and the protocols for proving them, are as follows:

[3]Observe that it would be possible to replace Equality 2 with a variant on Equality 1, namely $m_2 = m_1'$ or $m_2 = m_2'$. This variant, however, permits cheating in the case that $m_1 = m_2$. Moreover, it yields somewhat less efficient proofs.

Equality 1: $m_1 = m'_1$ **or** $m_1 = m'_2$.

Equality 2: $m_1 m_2 = m'_1 m'_2$.

The mix server demonstrates the Equality 1 using $\mathcal{DISPEP}$. For Equality 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.

4.3 Executing Millimix

Our description of Millimix is now simple. Mix servers agree in advance on a comparitor network architecture S. Then mix server 1 takes the ciphertexts $E = E_1, E_2, \ldots, E_n$ posted to the bulletin board and passes them through S, simulating each comparitor in turn as described in Section 1.1. He posts his equality proofs to the bulletin board, and all other mix networks verify them. If mix server 1 is caught cheating, then a majority coalition of the remaining mix networks decrypt his secret x_1 (which was shared among them at the beginning of the protocol), and remove him from the mix network. If mix server 1 produces verifiably correct output, then this is passed to mix server 2, who does the same. Each mix server i similarly simulates S on the output generated by the previous server until mix server k is reached. This final mix server posts his output $E' = E'_1, E'_2, \ldots, E'_n$ to the bulletin board, completing the mix network operation.

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] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* McGraw Hill, 1994.

[3] R. Cramer, I. Damgård, and B. Schoemakers. 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.

[4] A. de Santis and L. di Crescenzo and G. Persiano and M. Yung. On Monotone Formula Closure of SZK. 35th Annual Symposium on Foundations of Computer Science (FOCS) 1994, pp. 454–465.

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

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

[7] M. Jakobsson. Flash mixing. In *Proc. of 1999 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 83–89, 1999.

[8] M. Jakobsson and A. Juels. Mix and match: A new approach to secure multiparty computation. Manuscript.

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

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

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

[12] 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.

[13] 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.