# An Optimally Robust Hybrid Mix Network (Extended Abstract)

Markus Jakobsson and Ari Juels

RSA Laboratories
Bedford, MA, USA
{mjakobsson,ajuels}@rsasecurity.com

Abstract

We present a mix network that achieves efficient integration of public-key and symmetric-key operations. This *hybrid* mix network is capable of natural processing of arbitrarily long input elements, and is fast in both practical and asymptotic senses. While the overhead in the size of input elements is linear in the number of mix servers, it is quite small in practice. In contrast to previous hybrid constructions, ours has optimal robustness, that is, robustness against any minority coalition of malicious servers.

## 1 Introduction

A *mix network* is a cryptographic primitive that takes as input a sequence of ciphertexts and outputs the corresponding plaintexts in a random order. The main security goal of this procedure is to hide the correspondence between inputs and outputs from all participants (apart, of course, from the fact that players will recognize their own contributions). This property is referred to in mix network constructions as *privacy*. A certain degree of privacy can in principle be obtained by giving the list of input elements to a trusted server, who then performs some operation (such as decryption) and randomly permutes the results before outputting them. Naturally, however, in this procedure, the server in question knows the exact relationship between input and output elements. Most mix server contructions aim at a stronger form of privacy by distributing the process among a collection of servers. In this model, full privacy is achieved provided that no quorum of servers collude with one another. In most constructions a quorum consists of the majority of participating servers, but a variety of threshold structures are possible. Given no quorum of faulty or colluding servers, two other properties desirable in a mix network are *correctness* and *robustness*. A mix network is said to be robust if it produces output irrespective of server faults or failures. Correctness in a mix network is the property that the set of outputs from the mix network consists of plaintexts corresponding exactly to the set of ciphertext inputs.

Introduced by Chaum [3] as a primitive for privacy enhancement, mix networks have proven a powerful cryptographic tool for a diverse range of applications. One of the first such applications, for instance, is that of originator-anonymous e-mail [3, 28]. The idea here is for users to encrypt their e-mail messages, and then apply a mix network to the resulting batch of ciphertexts. The output of this mixing operation is the set of original plaintext e-mail messages. The privacy property of the mix network ensures that no one can determine which plaintext e-mail message corresponds to which ciphertext. Thus, even if it is known which user posted which ciphertext, the mix network in this application enables plaintext e-mail messages to be rendered anonymous. The literature includes a broad range of related applications of mix networks, including anonymized Web browsing [9] and secure elections [13, 27, 30], as well as seemingly unrelated applications such as anonymous payment systems [19] and general secure multiparty computation [18]. In this paper, we present a mix network tailored for applications that require very high throughputs of long messages, and where robustness is of importance. Such applications include private browsing and streaming, e-mail delivery, and privacy-preserving applications relating to advertisements [21].

### 1.1 Related work

We present a robust mix network that takes input ciphertexts of arbitrary (but equal) length and outputs the corresponding plaintexts. The mix network introduced by Chaum [3] and related proposals such as those in [29, 31] handle long inputs in a natural and efficient way, due to their intensive use of both public-key and symmetric-key encryption. The principle used in these schemes is that of iterated encryption. In a first step, the input plaintext is encrypted using the public key of the last mix server. Then, in a second step, the resulting ciphertext is encrypted using the public key of the second to last mix server. This is repeated until finally an encryption of the previous ciphertext, using the first mix server's public key is performed. These encryption steps are performed by the player who wishes to have the message (the plaintext above) output by the mix network. All of the encryption steps make use of probabilistic encryption [12], thereby preventing an attacker from matching input and outputs by applying the encryption function to outputs and matching the results against inputs. The final ciphertext that is submitted to the mix network, along with other ciphertexts generated in the same manner, potentially by different users. After all ciphertexts have been received, they are processed by the first mix server, who decrypts them all (using his secret key) and permutes the results be-

fore handing these to the second server, who in turn decrypts and permutes, etc. Finally, the last mix server decrypts, permutes, and outputs what will correspond to all the initial plaintexts, assuming everything went well. By straightforward enveloping techniques, i.e., combination of public-key and symmetric-key encryption, it is easy to see how plaintexts of arbitary length can be efficiently accommodated in this construction. We refer to this basic Chaumian mix network and other mix networks similarly amenable to such conmbination of public-key and symmetric-key techniques as *hybrid* mix networks.

What could – most notably – go wrong in the Chaumian mix network is that one of the mix servers replaces one or more partially decrypted ciphertexts. In a robust scheme, such a replacement attempt would not go unnoticed, and the remaining servers (all but those who were caught cheating) would replace the cheaters and re-execute the mixing operation. (This will be explained in more detail later.)

Subsequently proposed mix network schemes, known as *public-key mixes*, have focused on achieving robustness, typically through heavy reliance on public-key operations [1, 2, 7, 15, 16, 17, 24, 25]. At their best, these proposals enable robustness against any minority coalition of corrupt servers. Their drawback is that they are in general substantially less efficient than their hybrid predecessors. Indeed, on long inputs, such mix networks are very slow, requiring that input elements be divided into segments, each one of which is processed as an individual asymmetric ciphertext.

The approaches used to achieve robustness in public-key based mixes relate to so-called zero-knowledge proofs. Such proof allows a player to convince one or more other players that some relation between two or more elements holds, but without leaking information about the elements. In particular, it is possible to prove that a set of input elements correspond to a set of output elements, and that their relation is that of decryption (using the secret key corresponding to a particular public key) and permutation. This would be done without leaking either the secret key or the secret permutation. Note that this is crucial in a mix network. It is also of importance – to allow the replacement of cheaters – to be able to perform the desired actions of one mix server (that has been found to cheat) by collaboration between sufficiently many honest mix servers. In other words, the computation should be possible to distribute, while maintaining all desired security and privacy properties.

The difficulty in producing a robust hybrid mix lies in that one must combine the use of symmetric ciphers with distributed computation and proofs of correctness. Due to their structure, symmetric-key algorithms are inherently difficult to distribute, and entirely impractical to perform zero-knowledge proofs on. It is therefore a challenge to construct an efficient and secure mix protocol that is inherently distributed. Recent work by Ohkubo and Abe [26] demonstrates that it is possible to construct a mix network with both the robustness property and efficient use of symmetric-key encryption, using *duplication* of computational ability instead of *distribution* of the same. Their scheme, as a result of taking this approach, only achieves robustness against $\sqrt{n}$ corrupt servers out of a total of $n$. This is so since it is based on the architecture suggested in [7], in which each layer of decryption is performed by a quorum of participants, none of which – due to requirements of privacy – may participate in any other layer.

### 1.2 Our work

In this paper, we describe a hybrid mix network with *optimal robustness*, i.e., robustness against any coalition of fewer than $n/2$ servers. Our notion of robustness, however, is somewhat weaker than the standard notion: In our scheme it is possible for a corrupt server colluding with a corrupt user to modify the ciphertext element derived from the ciphertext of the corrupt user during one stage of the mix process. On the other hand, it is *not* possible for the colluding parties to modify elements derived from honest users; *nor* is it possible to modify any element *after* learning any portion of the output of the mix network. Thus, this beauty flaw amounts merely to making it possible for the adversary to "delay making up his mind", and it does not allow her to alter the distribution of the output, nor does it reduce the privacy of the scheme. For most practical purposes, this weaker notion of robustness is therefore benign.

Our scheme accepts ciphertexts derived from plaintexts of any polynomial size. The (additive) overhead associated with encryption of plaintexts is proportional to the number of encryption steps, equaling, in turn, the number of active mix servers, namely $n$. In practical terms, this expansion is marginal, and in particular for long inputs, which is the type of inputs on which hybrid encryption in general is well suited. Our scheme has per-server asymptotic worst-case and expected computational costs of $O(Nn^2)$ and $O(Nn)$ modular exponentiations respectively, where $N$ is the number of input elements to the mix, and $n$ is the total number of mix servers. (We disregard the cost of symmetric-key operations, which are generally small by comparison.) Our *worst-case* asymptotic cost is the same as the *expected* asymptotic cost of [26] when the costs are considered as functions of the number of mix servers the scheme is resilient against. Our *expected* costs – the costs incurred in the absence of an active adversary – are lower. Comparing our construction to some non-hybrid schemes, we see that it is not as efficient as some of these. In particular, it is not as efficient as the construction of Jakobsson [16], or the repaired version of this protocol by Mitomo and Kurosawa [24], whose asymptotic per-server costs in terms of the number of modular exponentiations are $O(N)$ in the expected and $O(Nn)$ in the worst case. For large $N$, the associated constants are quite low.[1] This cost analysis, however, assumes input elements short enough to be represented as a single ciphertext. Long input elements thus require either a naïve expansion of the modulus size or decomposition into many ciphertexts, with modifications made to the mix architecture (no techniques for which are actually described in the literature). Thus for long input elements, it may be expected that the concrete costs for the mix network proposed here are much lower.

As a first approach to building a hybrid mix, one might try to introduce robustness into the basic Chaum mix by appending a checksum to each layer of a given ciphertext input. Thus, when the $i^{th}$ server decrypts a layer for some message, it reveals a checksum, for which it can verify the correctness. If the $i-1^{st}$ server introduces naïve (or unintentional) errors into its output, these are likely to be detected. In particular, the $(i-1)^{st}$ server does not the checksum in the ciphertext being passed to the $i^{th}$ server, so it is unlikely to be able to determine a way of, e.g., flipping a few bits without being detected. On the other hand, the

[1] The assessment of these costs in [7, 24] are in terms of the number of modular arithmetic operations of any kind, including modular additions, and thus somewhat misleading.

$i^{th}$ server can simply *replace* a ciphertext in its output with an entirely new one, computing each layer and the associated checksums from scratch. Thus, this method of adding checksums does not in itself provide robustness.

In our construction, however, we employ a related idea. Instead of a checksum, we append a MAC to each layer in the mix. With this approach, we change the task of the attacker: now, in order to alter a message, the attacker must alter the corresponding MAC. To ensure robustness, our goal now resolves to that of protecting the MAC keys themselves. The central in our construction is to protect MAC keys by means of public-key-based encryption. In particular, the MACing key $k_i$, used by server $i$ to check the integrity of a given input, is encrypted in such a way that it is only available to server $i$ itself, and not to any of the other servers. Hence, none of the previous servers can determine $k_i$ or alter or replace ciphertexts so as to deceive server $i$.

This integrity protection on MAC keys is accomplished by proving that the *product* of MAC keys input to a given server is correct (or, more precisely, a public-key encryption thereof). Each server processes these encrypted MAC keys to extract the keys for its own use and to create new encryptions (of related MAC keys) for the next server. Note that if we prove that the correct relation holds between the product of two sets of MAC keys, such a proof is independent of the permutation applied to these keys and the associated messages. This simplifies the proof considerably, and is one of the critical elements enabling us to build an efficient mix construction.

It is important to note that is *is* possible for an attacker with control of one or more servers to modify a set of encrypted MAC keys so long as the product remains correct. Such an attacker can then replace a number of ciphertexts and compute correct MACs for these spurious ciphertexts. In order to cause the product of the full set of MAC keys to be correct, however, this attacker must introduce at least one MAC key that the attacker cannot feasibly choose itself. The attacker will be able to learn a ciphertext on this MAC key but, as we prove, cannot learn the key itself. In consequence, the attacker will with overwhelming probability produce an input to an honest server containing an incorrect MAC, and will thus be detected.

In presenting our new hybrid mix construction, we introduce and employ a few techniques of potential independent value. Two of these are sketched above: First, the use of MACs for purposes of correctness checks in multi-party protocols; second, the careful separation of the ability to verify a MAC and to perform decryption (which is important in order to allow one to be performed by a quorum, without allowing the other to also be performed as a result). We employ a third important technique in the last stage of the mix. In particular, we *simulate* the last server in the mix by means of distributed execution involving a quorum of participating mix servers. The purpose of this last step is to guarantee that no errors are introduced into the final output.

### Organization

We begin in section 2 by defining the desired properties of our scheme. In section 3, we describe the setup for our scheme and the underlying cryptographic primitives. We present our mix network construction and related claims in section 4. Due to lack of space, we omit a detailed security analysis from the body of this extended abstract, and refer to the extended version on the authors' webpages [14, 20] for proofs and proof sketches.

## 2 Definitions

As for previous mix networks, we have two types of participants: The *users* $\mathcal{U} = \{U_i\}_{i=1}^{N}$ (where we assume for simplicity that $U_i$ posts a unique input ciphertext $I_i$ to the mix); and the *servers* $\mathcal{S} = \{S_i\}_{i=1}^{n}$, who compute an output vector $\mathcal{O}$ from the input vector $\mathcal{I}$. The sets $\mathcal{U}$ and $\mathcal{S}$ may overlap. We consider security against a static adversary $\mathcal{A}$ that controls some proper subset $\hat{\mathcal{U}}$ of $\mathcal{U}$, and also a minority subset of $\mathcal{S}$, i.e., a subset of size at most $t$, denoted by $\hat{\mathcal{S}}$. All participants can be modelled by polynomial-time Turing Machines.

While we keep with the spirit of previous work in terms of our definitions for privacy and robustness, we modify these slightly to allow for active involvement by corrupt users $\hat{\mathcal{U}}$ in the mix protocol.

Let $\mathcal{I}$ be the set of inputs provided by $\mathcal{U}$, let $\hat{\mathcal{I}}$ be that portion provided by the corrupt users $\hat{\mathcal{U}}$, and let $\mathcal{I}'$ be the set provided by honest users, i.e., $\mathcal{I}' = \mathcal{I} - \hat{\mathcal{I}}$. Let $\mathcal{O}$ be the set of messages output from the mix network. (Note that there may be fewer outputs than inputs, as invalid inputs are eventually eliminated by the servers.)

We begin with the following, straightforward definition of *correct decryption* by a server $S_i$. Here, $D_k$ denotes symmetric decryption under key $k$.

**Definition 1** *We say that a triple $C' = (y', c', \mu')$ represents a* correct decryption *of triple $C = (y, c, \mu)$ by server $S_i$ if $c' = D_{k'}[c]$, where $k' = y^{\beta_i}$, $y' = y^{\alpha_i}$, and $\mu'$ equals $MAC_{y^{\gamma_i}}[c', I]$ for a nonce $I$. Here, $\alpha_i, \beta_i$, and $\gamma_i$ are private keys of server $S_i$.*

This definition means that $S_i$ has followed the decryption procedure in the protocol correctly for ciphertext $C$, as shown below. We extend the definition naturally to decryption by multiple servers. In other words, if $C$ represents input to $S_i$ and $C'$ represents output by $S_j$ for some $j \geq i$, we say that $C'$ represents a correct decryption of $C$ if it results in the obvious way from $C$ after a chain of correct decryptions. We also describe a group of ciphertext outputs from $S_j$ as representing a correct decryption of a group of ciphertext inputs to $S_i$.

Also important to our analysis is the somewhat unorthodox notion of *correct rendering*, defined as follows.

**Definition 2** *We say that a triple $C' = (y', c', \mu')$ represents a* correct rendering *of triple $C = (y, c, \mu)$ by server $S_i$ if $y' = y^{\alpha_i}$, and $\mu'$ equals $MAC_{y^{\gamma_i}}[c', I]$, for a nonce $I$, and where $\alpha_i$ and $\gamma_i$ are private keys of server $S_i$ as defined above.*

It is important to note how this definition is unusual. In the straightforward mix server execution, the pair $(c', \mu')$ represents a correct decryption of $c$ under the decryption key $y^{\beta_i}$, where $\beta_i$ is a private key of server $S_i$. Our term "rendering", however, implies that there may be a substitution for the straightforwardly decrypted value if accompanied by a correct MAC. Indeed, as stated before, a server in our protocol can collaborate with the user who posted an input to change it in mid-execution.

Let $\mathcal{I}_i$ be a set of input messages to a given mix server $S_i$. Let $\mathcal{O}_i$ represent the output of server $S_i$. (Note that for $i \leq n$, the set $\mathcal{O}_i$ will contain ciphertexts, in contrast to $\mathcal{O}$, which comprises plaintexts.) Let us extend the term "corresponds" in the obvious manner to these sets.

**Definition 3 (Correctness)** *We say that the output set $\mathcal{O}_i$ of server $S_i$ is* correct *with respect to $\mathcal{I}_i$ if the following hold:*
*(a) Let $\mathcal{I}'_i \subset \mathcal{I}_i$ be the full set of inputs containing correct decryptions of elements in $\mathcal{I}'$. Then there is a set $\mathcal{O}'_i \subset \mathcal{O}_i$ that contains a unique correct decryption of every element in $\mathcal{I}'_i$.*
*(b) Let $\hat{\mathcal{I}}_i$ represent the set of remaining inputs. There is a set $\hat{\mathcal{O}}_i$ that contains a unique correct rendering of every element in this valid input set.*
*(c) There are no other additional elements in $\mathcal{O}_i$.*

We extend this definition in the obvious manner to respective input and output sets $\mathcal{I}$ and $\mathcal{O}$ to define *correct* output for the full mix network. In particular, correct output includes unique correct decryptions of all the inputs from honest users, and unique correct renderings of all other valid inputs, where *unique* simply means that no duplicates are inserted.

As with traditional definitions of correctness, ours ensures that the adversary is unable to alter the inputs and corresponding outputs of honest players. Our definition of correctness, however, is unusual in two respects. First, by including the notion of "rendering", it allows for players to collude with servers to make substitutions during the protocol execution (as mentioned above). Second, our definition assumes the elimination of invalid input elements, while correctness definitions very often treat all inputs as valid.

Because of the notion of "rendering", correctness in our protocol does not guarantee robustness in the traditional sense. In particular, dishonest users may try to alter their inputs based on those of honest users, or dishonest users and servers may collude to change input or output values in the middle of a mix network execution based on input values and intermediate transcripts. In a payment scheme, for example, the adversary might try to rig dishonest submissions in the middle of the mix execution to duplicate honest submissions. Thus, in addition to requiring correctness, our robustness definition must ensure against the possibility of the adversary creating a correlation between the outputs of dishonest users and those of honest users. Our robustness definition ensures against this possibility in clause (c). Note that clause (c) is in fact critical for privacy as well as full robustness: if the adversary $\mathcal{A}$ is able to correlate inputs of dishonest users with those of honest users, she may be able to trace the inputs of honest users.

**Definition 4 (Robustness)** *A mix protocol is* robust *if, for input set $\mathcal{I}$, and in the presence of a static, active adversary $\mathcal{A}$ as described above:*
*(a) The protocol terminates in polynomial time in $N, n$ and all security parameters.*
*(b) The output of the protocol is* correct *with overwhelming probability over the coin flips of all participants.*
*(c) Let $\mathcal{O}'$ represent the set of plaintext outputs that are correct decryptions of $\mathcal{I}'$, and $\hat{\mathcal{O}} = \mathcal{O} - \mathcal{O}'$ the plaintext renderings of inputs from dishonest users. Then $\hat{\mathcal{O}}$ is computationally independent of the output plaintext set $\mathcal{O}'$ of honest users.*

Clause (c) may be stated more formally as follows. We consider the following experiment. The adversary chooses a pair $(O_0, O_1)$, where $O_0$ and $O_1$ are two distinct assignments to the plaintext set $\mathcal{O}'$ of honest users. A secret coin is then flipped to yield a bit $b$; the input set $\mathcal{I}'$ is selected uniformly at random to yield output plaintext set $O_b$. This set is them piecewise encrypted according to the method of encrypting plaintexts for the mix network, and the resulting ciphertexts are fed to the mix network, along with other input elements, submitted by the adversary. Now the adversary $\mathcal{A}$ participates in the execution of the mix network, but does not see $\mathcal{O}'$ (or the associated decryption and MAC keys). Clause (c) of our robustness definition is satisfied if no such adversary can guess $b$ with probability non-negligibly greater than $1/2$.

Finally, we consider the property of *privacy*. Our definition states informally that the adversary cannot determine which input elements provided by the honest user correspond to a given plaintext $m$ significantly better than by making a guess at random.

**Definition 5 (Privacy)** *We say that a mix network has the property of* privacy *against an active adversary $\mathcal{A}$ if the following holds. Let us suppose that the set of inputs $\mathcal{I}'$ provided by honest users contains $r$ instances of ciphertexts corresponding to plaintext $m$. Then the adversary is incapable with probability non-negligible greater than $r/|\mathcal{I}'|$ of finding an element $C \in \mathcal{I}'$ such that $C$ corresponds to $m$.*

## 3 Setup and building blocks

### 3.1 System parameters and setup

Public keys in our scheme are drawn from a cyclic group $\mathcal{G}$ of order $q$, for some large, published prime $q$. A typical choice would be a subgroup of order $q$ of $Z_p$, where $p$ is a large prime such that $q \mid (p-1)$. We let $g$ denote a published generator for $\mathcal{G}$. The security of the public-key operations in our scheme depends upon the hardness of the Decision Diffie-Hellman (DDH) problem over $\mathcal{G}$.

Our scheme involves the participation of an odd number $n = 2t + 1$ of servers, denoted $S_1, S_2, \ldots, S_n$. Let $Y_0 = g$. As a preliminary to the mixing operation, each server $S_i$ selects three private keys, $\alpha_i, \beta_i, \gamma_i \in_U Z_q$, where $\in_U$ denotes uniform, random selection. It then publishes a triple $(Y_i, K_i, Z_i)$ of corresponding public keys such that $Y_i = {Y_{i-1}}^{\alpha_i}$, $K_i = {Y_{i-1}}^{\beta_i}$, and $Z_i = {Y_{i-1}}^{\gamma_i}$. Each server proves knowledge of his private keys. Then, all three private keys of each server are distributed among the full set of servers using a $(t+1, n)$-threshold scheme. This key setup may be performed in a distributed manner using verifiable secret sharing (VSS) techniques. (We omit details, but refer the reader to [11] for an overview and important caveat.) Observe the somewhat unusual feature that the keys $(Y_i, K_i, Z_i)$ of server $S_i$ depend upon the key $Y_{i-1}$ of server $S_{i-1}$. While this dependence enforces a strict order on the key setup, it does not alter the basic techniques for accomplishing it. Servers additionally perform a joint, $(t+1, n)$-threshold generation of private keys $(\beta_{n+1}, \gamma_{n+1})$ for a simulated server $S_{n+1}$, with corresponding public keys $(K_{n+1}, Z_{n+1})$ using, e.g., techniques from [11]. Let $\alpha_{n+1,i}$ and $\gamma_{n+1,i}$ denote the respective shares of the private keys held by $S_i$.

We assume the existence of a *bulletin board*. This is a publicly shared piece of memory to which all players have read access and appendive, sequential write access with

authentication.[2] We assume further that all writes to the bulletin board proceed in synchronous time steps.

### 3.2 Public-key algorithms

**Proof of equivalence of discrete logs.** An important building block in our schemes is a protocol for proving of a quadruple $(a, b, y, z) \in \mathcal{G}^4$ that $\log_a b = \log_y z = x$, where the prover knows $x$. This may be accomplished using standard proof-of-knowledge techniques. In particular, the prover demonstrates knowledge of $\log_a b$ and $\log_y z$ relative to a common challenge $c$, as follows. She selects $r \in_U \mathcal{G}$, computes commitments $w_1 = a^r$ and $w_2 = y^r$, and sends these to the verifier. The verifier returns a challenge $c \in_U Z_q$. The prover provides response $s = cx + r \bmod q$. The verifier checks that $w_1 b^c = a^s$ and $w_2 z^c = y^s$. The verifier may, of course, consist of a coalition of servers, provided that the challenge is carefully generated. The scheme is honest-verifier zero-knowledge with soundness dependent on the discrete log problem. The protocol may be rendered non-interactive using the Fiat-Shamir heuristic [8]. In this case, $c$ is computed through application of a suitable hash function to $w_1$ and $w_2$, and security proofs depend additionally on the random oracle model. See [4, 5, 6] for further details. We denote a proof on the tuple $(a, b, y, z)$ by $EQDL[a, b, y, z]$.

**Compressed key scheduling.** We introduce and make use of an encryption method that we refer to as *compressed key scheduling*, and which is a generalization of a method recently and independently introduced in [26].

Our compressed key scheduling is essentially a public-key encryption scheme whereby a sender encrypts a set of (random) keys $\{k_i\}_{i=1}^{n+1}$ for all servers as a single ciphertext $y_0$. To do so, the sender selects a random exponent $\rho \in Z_q$, and constructs the ciphertext $y_0 = {Y_0}^{\rho}$. The set of keys $\{k_i\}_{i=1}^{n+1}$ is defined as $k_i = {K_i}^{\rho}$, for $1 \leq i \leq n+1$ and previously defined $K_i$. Observe that the sender may herself easily compute $\{k_i\}_{i=1}^{n+1}$.

To extract their respective keys, the servers do as follows. On receipt of $y_{i-1}$, server $S_i$ computes $(y_i, k_i, z_i) = ({y_{i-1}}^{\alpha_i}, {y_{i-1}}^{\beta_i}, {y_{i-1}}^{\gamma_i})$ which equals
$({Y_i}^{\rho}, {K_i}^{\rho}, {Z_i}^{\rho})$. Server $S_i$ then sends $y_i$ to server $S_{i+1}$. This enables server $S_{i+1}$ similarly to compute its keys. At the end of the protocol, each server $S_i$ has derived keys $y_i, k_i, z_i$, where the first is passed on to the next server and the second is used for decryption after having checked the correctness of the incoming ciphertext using the third one. We note that no coalition of fewer than $t+1$ servers, not including $S_i$, can feasibly learn the decryption key $k_i$.

Compressed key scheduling may be rendered robust by having each server $S_i$ post $y_i$ to the bulletin board, along with a proof of correct exponentiation $EQDL[y_{i-1}, y_i, Y_{i-1}, Y_i]$. In the case where server $S_i$ fails to publish $y_i$ correctly, a group of $t+1$ other servers can compute $y_i$ distributively without revealing the private keys of $S_i$. Given the similarity of techniques here to those involved in threshold signature schemes, such as those for DSS, we do not consider details here. We instead refer the reader to, e.g., [10].

[2] A bulletin board may be simulated or replaced by an authenticated broadcast channel or Byzantine agreement protocol [23]. In an asynchronous network, the latter is only robust against an adversary actively corrupting fewer than one-third of the servers, and alters the security of our mix construction accordingly.

It is possible to use more straightforward techniques to achieve essentially the same functionality as compressed key scheduling. The advantage of this new construction lies in its efficiency. First, the sender need only provide a single key for many servers. Second, as we shall see, it is possible to batch the associated $EQDL$ proofs in a manner that achieves a very high degree of computational and communication efficiency.

### 3.3 Symmetric-key algorithms

We employ a symmetric encryption scheme in our construction. Additionally, in order to defend against attacks in which previously posted ciphertexts are posted again or altered by malicious servers, we use a symmetric-key variation of the standard method (see [15]) of augmenting the ciphertext with a proof of knowledge, making this proof relative to the ciphertext and some session-specific nonce $I$.

**Message authentication code (MAC).** Let $k \in \mathcal{G}$ be a symmetric key shared by a sender and receiver.[3] We denote by $MAC_k[m]$ the message authentication code under key $k$ of message $m$ for any $m \in \{0,1\}^*$. We denote by $l_{MAC}$ a security parameter specifying the bit-length of the output of the MAC.

The essential security property we rely on for our construction is this. Suppose an adversary without any knowledge of $k$ is given message authentication codes $MAC_k[m_1], MAC_k[m_2], \ldots, MAC_k[m_u]$ for some $u$ that is polynomial in $l_{MAC}$. It is infeasible for the adversary to produce $MAC_k[m]$ on any message $m \notin \{m_i\}_{i=1}^{u}$. We explore this security requirement more formally in the appendix of the extended version [14, 20] of this paper.

**Symmetric-key encryption.** Again, let $k \in \mathcal{G}$ be a symmetric key shared by a sender and receiver. We denote by $E_k[m]$ the ciphertext on $m$ under key $k$, and by $D_k[c]$, the decryption of ciphertext $c$ under key $k$. Let $l_{enc}$ be a security parameter on the encryption scheme. We denote the cipher by $(E, D)$. We make use of the following *indistinguishability assumption* on the symmetric-key cipher for our scheme.

**Assumption 1 (Indistinguishability)** *Let the keys $k_0, k_1 \in_U \mathcal{G}$ be independently generated. Consider the following experiment. An adversary with resources polynomially bounded in $l_{enc}$ selects equal-length plaintexts $m_0, m_1 \in \{0,1\}^*$. For a random bit $b \in_U \{0,1\}$, the adversary is given ciphertexts $c_0 = E_{k_0}[m_b]$ and $c_1 = E_{k_1}[m_{1-b}]$. The adversary outputs a bit $b'$. The indistinguishability property states that there is no adversary such that $b' = b$ with probability $1/2 + \epsilon$ for positive $\epsilon$ non-negligible in $l_{enc}$.*

If $E$ is stream cipher based on a pseudo-random generator (PRNG), then this assumption may be based on the indistinguishability property of the PRNG. See [22] for a comprehensive treatment of PRNGs.

[3] Typically, one does not use an element $k \in \mathcal{G}$ as a MAC key in practice. It is easy, however, to convert $k$ to the more conventional form of symmetric key, such as a short bitstring. One possible means is appropriate application of a hash function to $k$.

## 4 Mix scheme

Our aim now is to fuse components described above in section 3 so as to combine the robustness of the underlying threshold public-key cryptosystem with the efficiency of the symmetric-key protocols. The central idea is to have players construct inputs to the mix network as sequences of concentric ciphertext layers, along with associated compressed key schedules. For a given input, each server $S_i$ derives the keys associated with the $i^{th}$ layer and removes it. Having removed the $i^{th}$ layer from all inputs, server $S_i$ passes the resulting ciphertexts to the next server.

The critical element in our construction is the ability of a server to prove that it has correctly removed a given layer. This is accomplished by having each ciphertext include a MAC in each encryption layer. In particular, the $(i+1)^{st}$ encryption layer of a given input includes a MAC employing a key derivable by server $S_{i+1}$ from the compressed key schedule. Since server $S_i$ does not have access to these MAC keys, it is infeasible for her to make substitutions or alterations to the mix elements without being detected by or colluding with server $S_{i+1}$.

Another useful method is what we call *open simulation*. A server is openly simulated by a quorum of players if these perform some or all of its computation, without disclosing the long-term secrets of the simulated player, but with no regard for the secrecy of temporary secrets (such as the permutation of values, if any.) Open simulation is used both to trace cheaters and to finish off the mixing process.

Yet another important element is the threshold distribution of the private keys of each server. In the case that server $S_{i+1}$ claims that server $S_i$ cheated, a coalition of the other servers can verify the proof sent from $S_i$ to $S_{i+1}$ and reconstruct the MAC keys of $S_{i+1}$ to verify her claim. The latter is performed using open simulation of the MAC verification done by $S_{i+1}$. Additionally, if any server $S_i$ fails to remove the $i^{th}$ encryption layer correctly from some ciphertext, the other servers can perform the removal (decryption) in a distributed manner, by open simulation of the decryption step of this player. Note that when a server complains, it need not reveal its state, and thus an honest server cannot have its private information extracted even if "sandwiched" by adjacent malicious servers. This is so since not all of a server's computation is openly simulated.

As the last server in this chain may itself be corrupt, we include a final step in which servers jointly verify the correctness of the final output, and decrypt those elements that are determined to be correct. (Note that this forces server $S_n$ to commit to his computation – by posting his output – before any server learns of the output plaintexts.) We may view this as the open simulation of a server $S_{n+1}$ responsible for authenticating the output of server $S_n$. This simulated server makes use of the private keys $(\beta_{n+1}, \gamma_{n+1})$ and the corresponding public keys $(K_{n+1}, Z_{n+1})$, which were jointly generated by the servers during the key generation phase of the protocol.

If any server is found to have cheated, he is expelled and simulated by a quorum of at least $t+1$ of the remaining servers, who are capable of reconstructing his private keys. The computation is rewound to openly simulate the execution of the cheating server in its entirely.

### 4.1 Concentric encryption

We begin by describing the algorithm used by a player to construct a ciphertext input to the mix network. This, the reader will recall, consists of a sequence of concentric layers of encryption, a concept first considered in its basic form in [3]. We refer to the encryption algorithm described here as *concentric encryption* and the resulting ciphertext as a *concentric ciphertext*.

At the beginning of a given mixing round, the servers jointly generate and publish a random nonce $I$ of length $l_{nonce}$. Additionally, they publish an integer $s$ describing the permitted length of plaintext inputs to the mix. (Shorter plaintexts may be padded out to $s$ bits.) These parameters are used in the generation of a concentric ciphertext.

**Input:** Plaintext $m \in \{0,1\}^s$.

**Output:** Ciphertext $(c_0, \mu_0, y_0)$. We refer to $c_0$ as the *base ciphertext*, $\mu_0$ as the *base MAC*, and $y_0$ as the *compressed key schedule*.

**procedure** Concentric Encrypt

1. *Compressed Key Schedule Generation.* The player selects a private key $\rho \in_U Z_q$. She computes

$$\begin{cases} k_i = {K_i}^{\rho} & 0 \le i \le n+1 \\ z_i = {Z_i}^{\rho} & 1 \le i \le n+1. \end{cases}$$

   She computes the compressed key schedule as $y_0 = {Y_0}^{\rho}$.

2. *Message Encryption.* The player encrypts the message $m$ by computing

$$\begin{cases} c_n = E_{k_{n+1}}[m] & \\ c_i = E_{k_{i+1}}[c_{i+1} \parallel \mu_{i+1}] & 0 \le i \le n-1 \\ \mu_i = MAC_{z_{i+1}}[c_i \parallel I] & 0 \le i \le n. \end{cases}$$

In the mix network protocol, players post equal-length input ciphertexts to the bulletin board until some triggering event occurs. For example, servers may set a predetermined limit on the number of input items to the mix, or else a deadline for the posting of input items. We denote the number of ciphertexts by $N$, and denote the ordered set of ciphertexts by $\{(c_0^{(j)}, \mu_0^{(j)}, y_0^{(j)})\}_{j=1}^{N}$.

### 4.2 Mix network for honest-but-curious servers

For ease of exposition, we first present a simplified mix network construction without robustness, but with privacy against an honest-but-curious adversary. Inputs are encrypted using the concentric encryption protocol presented above. Servers remove any duplicate inputs at the beginning of the protocol.

**Input:** Concentric ciphertext sequence $\{(c_0^{(j)}, \mu_0^{(j)}, y_0^{(j)})\}_{j=1}^{N}$ on equal-length plaintexts $\{m^{(j)}\}_{j=1}^{N}$.

**Output:** Plaintext sequence $\{m^{\pi(j)}\}_{j=1}^{N}$, for secret permutation $\pi$.

**Protocol** Honest Hybrid Mix

1. *Compressed Key Schedule Generation.* Server $S_i$ takes input $(c_{i-1}^{(j)}, \mu_{i-1}^{(j)}, y_{i-1}^{(j)})$, for $1 \leq i \leq N$, and computes its keys as follows for $1 \leq j \leq N$.

$$\begin{cases} \tilde{y}_i^{(j)} = (y_{i-1}^{(j)})^{\alpha_i} \\ \tilde{k}_i^{(j)} = (y_{i-1}^{(j)})^{\beta_i} \end{cases}$$

2. *Message Decryption.* Server $S_i$ performs the decryption:

$$(\tilde{c}_i^{(j)} \parallel \tilde{\mu}_i^{(j)}) \leftarrow D_{k_i^{(j)}}[c_{i-1}^{(j)}]$$

3. *Permutation.* $S_i$ randomly permutes the ordered set $\{(\tilde{c}_i^{(j)}, \tilde{\mu}_i^{(j)}, \tilde{y}_i^{(j)})\}_{j=1}^N$. In particular, the server $S_i$ selects a permutation $\pi_i$ on $N$ elements uniformly at random, and sets $(c_i^{(j)}, \mu_i^{(j)}, y_i^{(j)}) = (\tilde{c}_i^{(\pi_i(j))}, \tilde{\mu}_i^{(\pi_i(j))} \tilde{y}_i^{(\pi_i(j))})$. The set of decrypted and permuted triplets $\{(c_i^{(j)}, \mu_i^{(j)}, y_i^{(j)})\}_{j=1}^N$ is posted to the bulletin board.

Server $S_{n+1}$ is openly simulated by all the other servers. The output of server $S_{n+1}$ is taken here to be the output of the mix network. Note that the MACs are not used here because of the assumption that the adversary is strictly passive. Apart from the presence of the MACs and the use of key compression, this construction is somewhat similar in spirit to previous non-robust hybrid constructions, such as those described in [3, 31]. The privacy of the construction may be seen to depend upon two things. First, the indistinguishability property of the symmetric-key cipher, which ensures the privacy of the mix. Second, the DDH problem. In particular, an adversary should be unable to link $y_{i-1}^{(j)}$ with $y_i^{\pi_i(j)}$. An additional security element resides in the nonce $I$ the use of MACs. This ensures against re-use of ciphertext components, and ensures the non-malleability of posted ciphertexts.

### 4.3 Full protocol

We now present the full hybrid mix network construction, with robustness against any numbers of users and any minority coalition of corrupt servers. We begin by recalling that the private keys of each server are distributed among the other servers according to a $(t+1, n)$-threshold scheme. Thus, any coalition of $t+1$ servers can simulate the operations of a given server $S_i$ without the participation of $S_i$. This means that they can verify that server $S_i$ processed a given input item correctly by reconstructing the associated keys. Likewise, such a coalition of servers can remove server $S_i$ from the current invocation of the mix network by reconstructing all of its keys for the current batch of input items. We use these observations to achieve robustness in the full protocol Hybrid Mix, but do not describe the relevant protocols in detail, as they are fairly standard.

Another important element in the full protocol is the method server $S_i$ uses to prove that it has extracted the keys $\{y_i^{(j)}\}_{i=1}^N$ correctly. Recall from above that $y_i^{\pi_i(j)} = (y_{i-1}^{(j)})^{\alpha_i}$. Server $S_i$ could straighforwardly prove this equality for each $j$ using $EQDL$ were it not for the privacy requirements. Instead of using any of the methods of existing mix networks, we provide a new and more efficient solution. Let $P_i = \prod_{j=1}^N y_i^{(j)}$, and let server $S_i$ prove that $P_i = {P_{i-1}}^{\alpha_i}$. This is not sufficient in itself to demonstrate that $y_i^{\pi_i(j)} = (y_{i-1}^{(j)})^{\alpha_i}$ for all $j$. In combination with the checks afforded by symmetric-key operations in the mix, however, this *batch proof* method does indeed ensure that individual keys have been correctly extracted, as we show in the appendix of [14, 20].

In the following, we assume that should a mix server refuse to cooperate or produce incorrect outputs and thus be expelled, then a quorum of the other servers can simulate the absentee. For practical purposes, there would be a time limit associated with each step, after which an inactive server is considered absent. We also assume that any identical duplicates of input elements are removed before the beginning of the protocol. Finally, we note that $N$, the cardinality of the set of ciphertexts in the mix, may diminish as invalid ciphertexts are eliminated. For simplicity, we do not note this explicitly in our protocol description.

**Input:** Concentric ciphertext sequence $\{(c_0^{(j)}, \mu_0^{(j)}, y_0^{(j)})\}_{j=1}^N$ on equal-length plaintexts $\{m^{(j)}\}_{j=1}^N$.

**Output:** Plaintext sequence $\{m^{\pi(j)}\}_{j=1}^N$, for secret permutation $\pi$.

**Protocol** Hybrid Mix

1. Each server $S_i$ obtains input the ordered set $\{c_{i-1}^{(j)}, \mu_{i-1}^{(j)}, y_{i-1}^{(j)}\}_{j=1}^N$, and performs the following steps:

   (a) *Key Regeneration.* Server $S_i$ computes its keys as follows for $1 \leq j \leq N$.

$$\begin{cases} \tilde{y}_i^{(j)} = (y_{i-1}^{(j)})^{\alpha_i} \\ \tilde{k}_i^{(j)} = (y_{i-1}^{(j)})^{\beta_i} \\ \tilde{z}_i^{(j)} = (y_{i-1}^{(j)})^{\gamma_i} \end{cases}$$

   (b) *MAC verification.* Server $S_i$ verifies that $\mu_{i-1}^{(j)} = MAC_{\tilde{z}_i^{(j)}}[c_{i-1}^{(j)} \parallel I]$ for all $1 \leq j \leq N$. If the MAC is incorrect for any $j$, then server $S_i$ invokes the procedure Verify Complaint(i, j) detailed below.

   (c) *Message Decryption.* Server $S_i$ performs the decryption:

$$(\tilde{c}_i^{(j)} \parallel \tilde{\mu}_i^{(j)}) \leftarrow D_{\tilde{k}_i^{(j)}}[c_{i-1}^{(j)}]$$

   (d) *Permutation.* Server $S_i$ randomly permutes $\{(\tilde{c}_i^{(j)}, \tilde{\mu}_i^{(j)}, \tilde{y}_i^{(j)})\}_{j=1}^N$. In particular, $S_i$ selects a permutation $\pi_i$ on $N$ elements uniformly at random, sets $(c_i^{(j)}, \mu_i^{(j)}, y_i^{(j)}) = (\tilde{c}_i^{(\pi_i(j))}, \tilde{\mu}_i^{(\pi_i(j))} \tilde{y}_i^{(\pi_i(j))})$, and posts to the bulletin board the ordered set $\{(c_i^{(j)}, \mu_i^{(j)}, y_i^{(j)})\}_{j=1}^N$.

(e) *Batch proof of correctness of output keys.* Server $S_i$ proves the correctness of the set $\{y_i^{(j)}\}_{j=1}^N$, as follows. Server $S_i$ proves that $P_i = {P_{i-1}}^{\alpha_i}$ as $EQDL[P_{i-1}, P_i, Y_{i-1}, Y_i]$, where $P_i = \prod_{j=1}^N y_i^{(j)}$. If $S_{i+1}$ determines that the proof is incorrect, then $S_{i+1}$ invokes
Verify Complaint.

2. The output of $S_n$ is $\{(c_n^{(j)}, \mu_n^{(j)}, y_n^{(j)})\}_{j=1}^N$. Servers simulate server $S_{n+1}$ as follows. Players jointly compute $z_{n+1}^{(j)}$ for $1 \le j \le N$, and then check the MACs on all messages output by $S_n$. If the MAC for message $j$ is incorrect, then servers invoke the procedure Verify Complaint(n + 1, j), otherwise the servers jointly compute $k_{n+1}^{(j)}$ and $m^{(j)} = D_{k_{n+1}^{(j)}}[c_n^{(j)}]$.

The procedure Verify Complaint(i, j) is used to investigate a complaint made by server $S_i$ that the input triple $(c_{i-1}^{(j)}, \mu_{i-1}^{(j)}, y_{i-1}^{(j)})$ is invalid. By simulating processing of the message in question by $S_i$ and, if need be, by $S_{i-1}$, servers can determine which of the following three possibilities holds: (1) The complaint of server $S_i$ is invalid; (2) Server $S_{i-1}$ deviated from the protocol; or (3) The ciphertext was invalid as posted. The servers jointly expel any corrupt server from the protocol or else remove the input triple from the mix if it is determined to be invalid. If a server is expelled, a replacement server is selected from a pool of players that have not yet been involved in the mixing (but only voting). If a triple is removed, it is purged from all previous steps (which can be done by each server revealing what input it corresponded to, and revealing the corresponding keys for verification purposes). The value $N$ is modified accordingly.

**Procedure** Verify Complaint(i, j)

Servers compute $\tilde{z}_i^{(j)}$ using their shares of $\gamma_i$;
If $\mu_{i-1}^{(j)} = MAC_{\tilde{z}_i^{(j)}}[c_{i-1}^{(j)} \parallel I]$ and the proof $EQDL[P_{i-2}, P_{i-1}, Y_{i-2}, Y_{i-1}]$ is correct, then
    Servers expel $S_i$ ("false alarm");
else if $i = 1$, then
    Servers remove $(c_0^{(j)}, \mu_0^{(j)}, y_0^{(j)})$ from the mix ("bad input");
else
    Server $S_{i-1}$ publishes $j' = \pi_{i-1}^{-1}(j)$;
    If $j' \notin \{1, 2, \ldots, N\}$, then
        Servers expel and simulate $S_{i-1}$ ("cheater");
    else
        Servers compute $k_{i-1}^{(j')}$ from $y_{i-2}^{(j')}$ using their shares of $\beta_{i-1}$;
        If $(\tilde{c}_{i-1}^{(j)} \parallel \tilde{\mu}_{i-1}^{(j)}) \neq D_{k_{i-1}^{(j')}}[c_{i-2}^{(j')}]$ ("incorrect decryption") or $\tilde{\mu}_{i-1}^{(j')} \neq MAC_{\tilde{z}_i^{(j')}}[c_{i-1}^{(j')} \parallel I]$ or $EQDL[P_{i-2}, P_{i-1}, Y_{i-2}, Y_{i-1}]$ is wrong ("should have complained"), then
            Servers expel and simulate $S_{i-1}$ ("cheater");
        else
            Servers remove $(c_{i-1}^{(j)}, \mu_{i-1}^{(j)}, y_{i-1}^{(j)})$ from the mix ("bad input").

**Remark on broadcast assumptions.** It will be observed that prior to the simulation of $S_{n+1}$, if servers sign their outputs, the protocol does not require the use of broadcast until and unless servers misbehave. The simulation of $S_{n+1}$ can in fact be modified to enable a similar elimination of broadcast assumptions. We briefly sketch the idea here. When it has finished its computation, server $S_n$ sends $\{c_n^{(j)}, \mu_n^{(j)}, y_n^{(j)}\}_{j=1}^N$ to all servers. Each server $S_i$ in turn sends $\{(z_n^{(j)})^{\gamma_{n+1,i}}\}_{j=1}^N$ to all other servers, along with non-interactive proofs that these shares are correctly constructed. This enables servers to compute $\{z_{n+1}^{(j)}\}_{j=1}^N$ through LaGrange interpolation, after which the MACs can be verified. Servers may similarly compute $k_{n+1}^{(j)}$ for all ciphertexts $c_n^{(j)}$ with valid MACs, allowing decryption of these ciphertexts. If, at any point in the protocol, some server does not send its results prior to an established time-out, or else some server detects an error, then it becomes necessary to make use of a broadcast channel.

### 4.4 Protocol efficiency and security

Let us briefly describe the asymptotic efficiency of Hybrid Mix, assuming that all proof protocols are non-interactive – and thus with security dependent on both the DDH and random oracle assumptions. In the optimistic case, i.e., assuming honest behavior by all servers, each server must perform computation equivalent to $\mathcal{O}(N + n)$ modular exponentiations and $\mathcal{O}(Nn)$ modular multiplications as a total for all inputs. In the presence of malicious behavior, costs rise to $\mathcal{O}(Nn)$ modular exponentiations per server. These tallies exclude the costs of symmetric-key operations. Note that it is the batch verification procedure that renders the optimistic costs lower than those for the malicious case. The aggregate broadcast complexity is $\mathcal{O}(snN)$ bits plus $\mathcal{O}(Nn)$ group elements for both the optimistic and malicious cases, where $s$ is the length of the plaintexts corresponding to the inputs.

A drawback to our construction is the cost of constructing an input ciphertext by means of
Concentric Encrypt. This requires $\mathcal{O}(n)$ modular exponentiations, in addition to the cost of the symmetric-key operations.

Our protocol is *robust* and also *private* according to our definitions in section 2.

## 5 Open Problems

It remains to be seen how to achieve public verifiability for an efficient hybrid mix. While this is theoretically straighforward using general multi-party computation techniques, neither [26] nor this paper succeeds in reaching this goal without invocation of such methods. It is the authors' belief that careful use of digital signatures, rather than MACs, may in fact enable public verifiability to be achieved within the protocol framework outlined here.


### Acknowledgments

The authors wish to thank Masayuki Abe, Ran Canetti, and Phil MacKenzie for their very helpful suggestions and feedback.

## 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 permutation networks. In K.Y. Lam, C. Xing, and E. Okamoto, editors, *ASIACRYPT '99*, pages 258–273, 1999. LNCS no. 1716.

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

[4] D. Chaum and T.P. Pedersen. Wallet databases with observers. In E.F. Brickell, editor, *CRYPTO '92*, pages 89–105. Springer-Verlag, 1992. LNCS no. 740.

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

[6] A. de Santis, G. di Crescenzo, G. Persiano, and M. Yung. On monotone formula closure of SZK. In *FOCS '94*, pages 454–465. IEEE Press, 1994.

[7] Y. Desmedt and K. Kurosawa. How to break a practical mix and design a new one. In B. Preneel, editor, *EUROCRYPT '00*, pages 557–572. Springer-Verlag, 2000. LNCS no. 1807.

[8] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In J. L. Massey, editor, *EUROCRYPT '86*, pages 186–194. Springer-Verlag, 1986. LNCS no. 263.

[9] 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.

[10] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In U. Maurer, editor, *EUROCRYPT '96*, pages 354–371. Springer-Verlag, 1996. LNCS no. 1070.

[11] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. The (in)security of distributed key generation in dlog-based cryptosystems. In J. Stern, editor, *EUROCRYPT '99*, pages 295–310. Springer-Verlag, 1999. LNCS no. 1592.

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

[13] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In B. Preneel, editor, *EUROCRYPT '00*, pages 539–556. Springer-Verlag, 2000. LNCS no. 1807.

[14] M. Jakobsson. Personal homepage. http://www.markus-jakobsson.com.

[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 *PODC '99*, pages 83–89. ACM, 1999.

[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 function evaluation via ciphertexts. In T. Okamoto, editor, *ASIACRYPT '00*, pages 162–177, 2000. LNCS No. 1976.

[19] M. Jakobsson and D. M'Raïhi. Mix-based electronic payments. In E. Tavares S and H.Meijer, editors, *SAC '98*, pages 157–173. Springer-Verlag, 1998. LNCS no. 1556.

[20] A. Juels. Personal homepage. http://www.ari-juels.com.

[21] A. Juels. Targeted advertising and privacy too. In D. Naccache, editor, *RSA Conference Cryptographers' Track*, 2001. To appear.

[22] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Univ. Press, 1996.

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

[24] M. Mitomo and K. Kurosawa. Attack for flash mix. In T. Okamoto, editor, *ASIACRYPT '00*, pages 192–204, 2000. LNCS No. 1976.

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

[26] M. Ohkubo and M. Abe. A length-invariant hybrid mix. In T. Okamoto, editor, *ASIACRYPT '00*, pages 178–191, 2000. LNCS No. 1976.

[27] C. Park, K. Itoh, and K. Kurosawa. All/nothing election scheme and anonymous channel. In T. Helleseth, editor, *EUROCRYPT '93*, pages 248–259. Springer-Verlag, 1993. LNCS No. 765.

[28] A. Pfitzmann and B. Pfitzmann. How to break the direct RSA-implementation of MIXes. In J.-J. Quisquater and J. Vandewalle, editors, *EUROCRYPT '89*, pages 373–381. Springer-Verlag, 1989. LNCS No. 434.

[29] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-MIXes: Untraceable communication with very small bandwidth overhead. In *Info. Security, Proc. IFIP/Sec'91*, pages 245–258, 1991.

[30] 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.

[31] P. Syverson, D. Goldschlag, and M. Reed. Anonymous connections and onion routing. In *Proc. of 18th Annual Symposium on Security and Privacy*, pages 44 – 54. IEEE Press, 1997.