RZ 3125 (# 93171) 05/03/99
Computer Science/Mathematics 11 pages

Research Report

Commitment-Based Fault-Removing MIX cascade

Ceki Gülcü

IBM Research Division
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Research Division
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

Commitment-Based Fault-Removing MIX cascade

Ceki Gülcü

IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

We present a new anonymous communication channel based on MIXes. The channel is such that faulty nodes can be identified and removed without violating the untraceability of the users, under the assumption that at least a single mix is honest and networking delays are bounded.

Our protocol has three phases, a key sharing phase, a commitment phase, and a sending phase. Once the first two phases are completed, the sending phase can be repeated to send multiple messages.

Our system has the advantage of being computationally more efficient than existing fault-tolerant cascades, in which each MIX has to perform in the order of 50 exponentiations per sender. On the other hand, the protocol presented here lacks a provision for fault compensation without restarting the protocol.

Our scheme makes use of the most basic cryptographic primitives, namely hash functions, signatures, and shared and public key encryptions.

1 Introduction

The MIX cascade introduced by Chaum in [2] in 1981 allows a user to send an e-mail message to a recipient without revealing the identity of the sender. Pfitzmann *et al.* [11] have extended Chaum's work to allow interactive (bi-directional) untraceable communication. Syverson *et al.* [14] have actually implemented a real-time anonymous channel based on similar techniques.

These systems are best effort in the sense that (arbitrarily) failing nodes will cause messages to be lost. Moreover, the channel will provide no information on how or why a message was lost. Thus, it is impossible - or at least very difficult - to unequivocally identify the faulty and possibly malicious node.

Robustness is particularly important when the untraceable channel is used as a primitive in voting schemes or in payment systems where loss of messages might be intolerable.

There have been proposals [12, 1, 9, 4] to solve the dependable untraceable channel problem. However, these resort to rather expensive zero-knowledge proofs.

In his seminal paper [2], Chaum proposes that each user should get a signed receipt for its submission and that the batch-output of each MIX be signed. Thus, when a MIX fails to process a message properly it can be prosecuted by the originator. However, to be able to prove wrongdoing, the user has to forsake anonymity. In many applications it is unacceptable to trade privacy for reliability.

We introduce a fault-removing [6] untraceable channel—guaranteeing untraceability (of the sender) at all times, even after a disruption or at fault-diagnosis time.

Real-time untraceable channels [11, 14] are generally composed of two distinct phases: a key-sharing phase (using public key cryptography) and an actual sending phase (using the previously shared keys). Our scheme adds an intermediate step in which MIXes commit to the shared keys and their permutations. These commitments are then certified by the users before the sending phase starts. This seemingly innocuous step decouples the inputs of the various users so that each disruption can be prosecuted individually while still maintaining protection of privacy. We shall come back to this later.

1.1 Cryptographic Assumptions

We denote by $e_K(m)$ the encryption of message m under shared key K. In addition to the usual confidentiality properties under the computational security model, we assume that the encryption function e_K possesses the following property:

Indistinguishability Without knowing keys K_1 and K_2, given ciphertexts $y_1 = e_{K_1}(x_1)$ and $y_2 = e_{K_2}(x_2)$ and corresponding cleartexts x_1 and x_2 (in random order), it is infeasible to correlate one of the ciphertexts to one of the cleartexts with probability of more than one half.

Let $i(x) : \{0,1\}^* \rightarrow \{0,1\}^t$, for some integer t, be a hash function. The decryption of ciphertext y is said to pass the *integrity check* if the decryption of y yields $x \mid\mid r$ such that $r = i_K(x)$, where $\mid\mid$ denotes string concatenation. We shall write $\{x\}_K$ to denote the redundant encryption $e_K(x \mid\mid i(x))$, namely the encryption of x and its associated redundancy under key K. We assume that this composition of the encryption function e_K and the hash function is such that:

Integrity Assumption Without knowing the shared key K, it is computationally infeasible to generate a ciphertext that foils the integrity check.

We assume that all participants in the protocol have long-term public keys. We write $\mathsf{E}_A(m)$ for the encryption of m under A's public key. The public key encryption function E is required to have the following property:

Indistinguishability Let $y_1 = \mathsf{E}(r_1 \| x_1)$ and $y_2 = \mathsf{E}(r_2 \| x_2)$, where r_1 and r_2 are random strings of "sufficient" length. Without knowing r_1, r_2 and the private key, given ciphertexts y_1 and y_2 and corresponding cleartexts x_1 and x_2 (in random order), it is infeasible to correlate one of the cypertexts to one of the cleartexts with probability of more than one half.

It is usually the case that encryption functions have unique decryptions (injectivity). The following property follows:

Target Collision-Resistance Given ciphertext $y = e_K(x)$, it impossible to find cleartext $x' \neq x$ such that $y = e_K(x')$, even if K is known.

Target collision-resistance implies that revealing the shared key K convinces any verifier that x is the correct decryption of y, if and only if $y = e_K(x)$. The assertion that K is the correct key must be proved

Table 1: Notation Summary

Notation	Explanation
$m_1 \| m_2$	concatenation of m_1 and m_2
$\mathsf{E}_A(m)$	encryption of m with A's public key
$\mathsf{sign}_A(m)$	A's signature on m
$\{m\}_K$	redundant encryption of m under shared key K
$\mathsf{com}[m]$	commitment to string m
δ_l	the composition of l permutations $\pi_l \circ \pi_{l-1} \circ \cdots \circ \pi_1$
ρ_l	the inverse of l permutations, that is $\rho_l = \delta_l^{-1}$

independently. The same argument holds for public key encryption except that the encryption function is public from the start.

We write $\mathsf{sign}_A(m)$ for A's signature on some message m, which by convention includes the message m. The signature scheme must be resistant against at least the chosen-message attack [7].

We shall make extensive use of public commitments. The commitment scheme employed should have the following properties:

Hiding If Alice commits to some value m randomly chosen from a large set, then a computationally-bounded Bob cannot learn m without the help of Alice.

Binding Computationally-bounded Alice cannot change the value m when asked to open the commitment.

Non-interactive Verifiability If Bob already knows m, then Bob can check that Alice committed to the correct value without having to interact with Alice.

We suppose that the above commitment scheme is implemented using a collision-resistant hash function [13]. We write $\mathsf{com}[m]$ to specifically refer to this hash function-based commitment on message m.

2 Construction

The protocol that we are about to present runs among a set $\mathcal{U} = \{P_1, \ldots, P_N\}$ of N users wishing to transmit untraceable messages and a set of L MIXes $\{M_1, \ldots, M_L\}$. User P_n knows that it is the n^{th} participant in the protocol. So does everybody else.

In the following, we shall use n as an index for users, running from 1 to N. Similarly, l shall be used as an index for MIXes, running from 1 to L.

Let Π_N denote the set of all possible permutations of the set $\{1, \ldots, N\}$. We write δ_l for the composition of l permutations, that is, $\delta_l = \pi_l \circ \pi_{l-1} \cdots \circ \pi_1$, with $\pi_i \in \Pi_N$ and $1 \leq i \leq l$. We shall often make use of the inverse permutation, $\rho_l = \delta_l^{-1}$.

Our failure model allows any node, be it a user or a MIX, to fail arbitrarily, in particular maliciously. Moreover, nodes can collude in order to defeat the protocol.

2.1 Phase One : Key Sharing

The following sections assume a fault-free situation. We shall defer the discussion on fault-treatment until Section 3.

At the end of the key sharing phase, unless disrupted, each participant in P_n will share a key K_n^l with each MIX M_l. The MIXes, except for the first, will not learn which key is owned by which user.

We shall write $\mathcal{O}_1$ to denote the set of entities observing the progress of phase one. The set $\mathcal{O}_1$ can consist of the empty set or anyone who cares to trace the progress of phase one.

Each user P_n begins by randomly choosing a key K_n^l that it will share with M_l. It then prepares an *onion* defined by

$$s_n^L = \mathsf{E}_{M_L}(K_n^L) \quad (1)$$

$$s_n^l = \mathsf{E}_{M_l}(K_n^l \| s_n^{l+1}) \quad \text{with } l = L-1, \ldots, 1. \quad (2)$$

Each user P_n signs the resulting onion s_n^1, and sends it to M_1 and $\mathcal{O}_1$. When all users have sent their onion, M_1 verifies each of the signatures and checks that the set of signers matches the set of users.

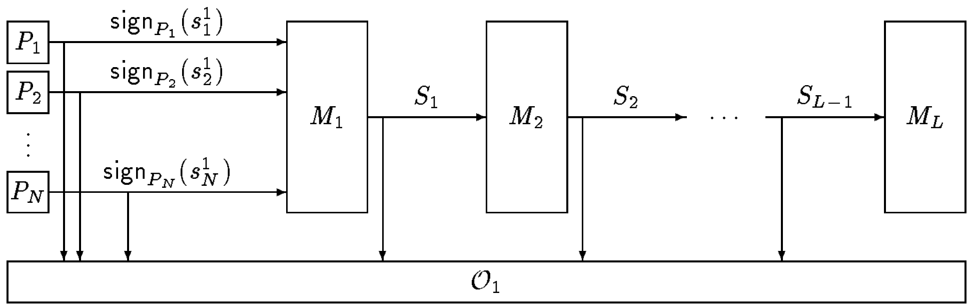

Figure 1: Message flows in phase one.

Then, M_1 proceeds to strip a layer of encryption from each of the onions s_n^1. The decryption yields K_n^1 and s_n^2.

At this point, M_1 chooses a permutation $\pi_1 \in_R \Pi_N$ and permutes all onions s_n^2 accordingly. The permuted results are concatenated and signed. Let S_1 denote this message, that is,

$$S_1 = \mathsf{sign}_{M_1}\big(s^2_{\rho_1(1)}||\cdots||s^2_{\rho_1(N)}\big), \tag{3}$$

where ρ_1 is the inverse of $\delta_1 = \pi_1$.

The composite message S_1 is sent to the next MIX M_2 and to $\mathcal{O}_1$.

The procedure for M_l, for $1 < l < L$, is quite similar. From the preceding MIX M_{l-1} it receives S_{l-1}, given by

$$S_{l-1} = \mathsf{sign}_{M_{l-1}}\big(s^l_{\rho_{l-1}(1)}||\cdots||s^l_{\rho_{l-1}(N)}\big), \tag{4}$$

where ρ_{l-1} is the inverse of $\delta_{l-1} = \pi_{l-1} \circ \pi_{l-2} \circ \cdots \circ \pi_1$. M_l verifies the signature on S_{l-1}. It then removes a layer of encryption from each s_n^l. The decryption yields K_l^n and s_n^{l+1}.

At this point, M_l permutes all onions $s^{l+1}_{\delta_{l-1}(n)}$ according to $\pi_l \in_R \Pi_N$. The permuted onions are concatenated and the result signed. Let S_l denote this message, that is,

$$S_l = \mathsf{sign}_{M_l}\big(s^{l+1}_{\rho_l(1)}||\cdots||s^{l+1}_{\rho_l(N)}\big). \tag{5}$$

The composite message S_l is sent to the next MIX M_{l+1} and to $\mathcal{O}_1$.

The last MIX, M_L, follows the same procedure except that it does not compose a message S_L (nor forward it to anyone). Figure 1 illustrates the message flows in phase one.

As shall be apparent later, at all three phases of the protocol, all MIXes, except the first, process one composite input message and produce one composite output message. Misusing the language to some extent, we will henceforth denote by i^{th} input (or output), the i^{th} component of the composite input (or output).

2.2 Phase Two : Commitment

At the end of this phase, unless disrupted, each MIX will commit to its permutation and its shared keys. More precisely, each MIX will commit to the key it will apply to a certain input and at which position it will output the (decrypted) result. Thus, MIX M_1 will output a batch-commitment C_1 defined as follows:

$$\begin{aligned} C_1 = \mathsf{sign}_{M_1}\Big(&\mathsf{com}\big[\pi_1^{-1}(1)||K^1_{\rho_1(1)}\big]\,\|\cdots \\ &\cdots\|\mathsf{com}\big[\pi_1^{-1}(N)||K^1_{\rho_1(N)}\big]\Big) \end{aligned} \tag{6}$$

For example, the i^{th} commitment $\mathsf{com}\big[\pi_1^{-1}(i)||K^1_{\rho_1(i)}\big]$ indicates that input $\pi_1^{-1}(i)$ will be output at the i^{th} position and that it will be decrypted with key $K^1_{\rho_1(i)}$.

Similarly, M_l's batch-commitment C_l is defined as

$$\begin{aligned} C_l = \mathsf{sign}_{M_l}\Big(&\mathsf{com}\big[\pi_l^{-1}(1)||K^l_{\rho_l(1)}\big]\,\|\cdots \\ &\cdots\|\mathsf{com}\big[\pi_l^{-1}(N)||K^l_{\rho_l(N)}\big]\Big). \end{aligned} \tag{7}$$

The set of observers for the commitment phase, $\mathcal{O}_2$, *must include* $\mathcal{U}$, the set of all N users. Each MIX M_l will send its output C_l to $\mathcal{O}_2$.

By Eq. (6), the m^{th} output in C_1 is written $\mathsf{com}\big[\pi_1^{-1}(m)||K^1_{\rho_1(m)}\big]$. Let $\pi_1(n) = m$, and consequently $\pi_1^{-1}(m) = \rho_1(m) = n$. Inserting accordingly, we obtain

$$\mathsf{com}\big[\pi_1^{-1}(m)||K^l_{\rho_1(m)}\big] = \mathsf{com}\big[n||K^1_n\big].$$

Knowing n and K^1_n, user P_n can compute $\mathsf{com}\big[n||K^1_n\big]$. By observing the position of this string in batch-commitment C_1, user P_n can learn the value of $\delta_1(n) = \pi_1(n) = m$.

Assume that, having observed $C_1, \ldots, C_{l-1}$, user P_n can learn $\delta_{l-1}(n)$. Knowing $\delta_{l-1}(n)$, user P_n can compute $\mathsf{com}\big[\delta_{l-1}(n)||K^l_n\big]$. Noting the position of this string in batch-commitment C_l yields $\delta_l(n)$. Inductive application of this technique on each of C_i yields $\delta_i(n)$ for $i = 1, \ldots, L$.

After checking that all the batch-commitments are in agreement with the keys it distributed in phase one, each user P_n will sign a message to attest its satisfaction with the outcome of phase two.

Although each user P_n learns $\delta_1(n), \ldots, \delta_L(n)$, it cannot learn anything on $\delta_1(m), \ldots, \delta_L(m)$ for $m \neq n$—except that they are not $\delta_1(n), \ldots, \delta_L(n)$.

Note that phase one is sequential with respect to the MIXes, whereas phase two can be performed in any order. For the sake of clarity, we chose to have two separate phases. However, in an actual implementation, phases one and two can be merged to reduce communication complexity.

The careful reader will have noted that seeing the outputs of phase one, user P_n can easily deduce $\delta_1(n), \ldots, \delta_L(n)$. In that case the commitment expression could be made more graceful. The commitment expression proposed here is specifically designed such that the users can verify a batch-commitment with only one hash function computation (com) even if they have not observed phase one.

2.3 Phase Three : Sending

At the end of this phase, unless disrupted, each user will send an untraceable message to its correspondent.

Let m_n denote the message that user P_n wishes to send to an arbitrary user, not necessarily in $\mathcal{U}$, located at $\mathcal{A}_n$. Although message m_n and the identities of P_n and $\mathcal{A}_n$ are revealed by our protocol, the relation between P_n and $\mathcal{A}_n$ is not.

To privately send message m_n to $\mathcal{A}_n$, each user P_n will multiply encrypt its message, once for each MIX on the path, as follows:

$$o^L_n = \big\{\mathcal{A}_n||m_n\big\}_{K^L_n} \tag{8}$$

$$o^l_n = \big\{o^{l+1}_n\big\}_{K^l_n} \quad \text{with } l = L-1, \ldots, 1. \tag{9}$$

The resulting onion o^1_n is *signed* by P_n and sent to the first MIX, namely M_1, and to $\mathcal{O}_3$, the observers of the sending phase. The set $\mathcal{O}_3$ may consist of the empty set.

When all users have sent their onions, M_1 will decrypt the messages with the corresponding shared keys. The MIX also checks that all decryptions pass the integrity check.

Should all decryptions satisfy the integrity check, M_1 permutes the results according to π_1. The permuted results are concatenated and then signed. Let O_1 represent this composite message, that is,

$$O_1 = \mathsf{sign}_{M_1}\big(o^2_{\rho_1(1)}||\cdots||o^2_{\rho_1(N)}\big). \tag{10}$$

Message O_1 is transfered to the next MIX M_2 as well as to $\mathcal{O}_3$.

Subsequent MIXes M_l, for $1 < l < L$, perform the same set of operations but with their respective keys and permutations. Each one receives O_{l-1}, given by

$$O_{l-1} = \mathsf{sign}_{M_{l-1}}\big(o^l_{\rho_{l-1}(1)}||\cdots||o^l_{\rho_{l-1}(N)}\big) \tag{11}$$

from the preceding MIX M_{l-1}. It verifies the signature on O_{l-1}. It then removes a layer of encryption from each o^l_n, yielding o^{l+1}_n. The results are permuted according to π_l, concatenated and signed. Let O_l represent this composite message, that is,

$$O_l = \mathsf{sign}_{M_l}\big(o^{l+1}_{\rho_l(1)}||\cdots||o^{l+1}_{\rho_l(N)}\big). \tag{12}$$

Message O_l is sent to the following MIX M_{l+1} as well as to $\mathcal{O}_3$.

The last MIX operates only slightly differently. It discovers the actual messages m_n and the destination addresses $\mathcal{A}_n$. It transmits the messages according to the permutation π_L. Thus, the first message it will send will be $m_{\rho_L(1)}$ to $\mathcal{A}_{\rho_L(1)}$, then $m_{\rho_L(2)}$ to $\mathcal{A}_{\rho_L(2)}$, and so forth. Figure 2 illustrates the message exchanges of phase three.

The sending phase can be repeated to send multiple messages without needing to reiterate the preceding two phases. In that case, the address information $\mathcal{A}_n$ in o^L_n may be omitted.

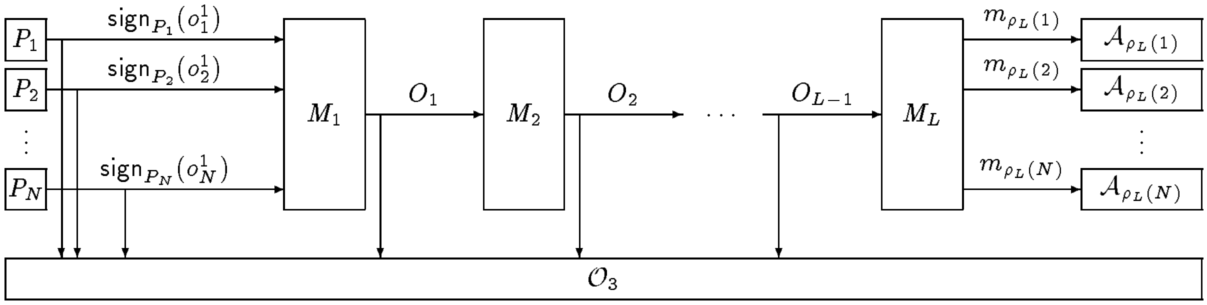

Figure 2: Message flows in phase three

2.4 Word of Caution

The reader should realize that the above construction would be easily defeated if implemented as is.

If the inputs to the MIX cascades are of different sizes, it is trivial for the attacker to link the different inputs to the different outputs based on message size. The various padding techniques developed in [2, 3] would thwart such size-based correlation.

The above construction neither assigns any unique identifier to the protocol, protocol version, step or protocol instance nor limits the scope of public keys, thus infringing on important rules of secure system design [5]. An attacker can conveniently take advantage of our "apathy." For example, as MIXes will basically decrypt anything thrown at them, nothing prevents an attacker from submitting a message encrypted with the attackers public key and hereby trivially obtaining a decryption.

Neither do we attempt to prevent replay attacks. For example, a dishonest MIX can disrupt the protocol at phase three and later replay a signed batch-commitment from a previous run of the protocol, egregiously clearing itself from any offense.

We have intentionally chosen to ignore these simple but important attacks to keep our description concise. However, an implementor will not benefit from the same luxury.

3 Fault Removal

In this section, we will show how failing nodes (users or MIXes) can be identified without relinquishing untraceability. This result holds under the assumption that *at least two*[1] MIXes are honest and that network delays are bounded.

We distinguish between involuntary and malicious failures. A *disrupter* is defined to be a malicious attacker injecting messages that cause the protocol to fail either immediately or at some later stage.

Our fault diagnosis paradigm is quite simple:

At any of the three phases, we allow the observers (of any phase) or the consumer of an output (if any) to claim that an output is erroneous. The party making the claim is called the *plaintiff*, the producer of the purportedly erroneous output is called the *defendant*.

Whenever an accusation is made, the defendant attempts to prove its innocence. This is done by the defendant showing that it has performed the correct operation on its input. If the defendant node cannot prove its innocence, it is labeled a disrupter. However, if it can prove that it is innocent, then the burden of proof shifts to the predecessor of the defendant. This is repeated until a disrupter has been found. However, if all parties are able to demonstrate their innocence, then our conclusions will depend on the actual phase of the protocol.

Henceforth, we shall employ *left* as a synonym for predecessor and *right* as a synonym for successor. For example, we shall write "to the left" to designate a predecessor of a MIX. Similarly, we shall say "to the right" to designate a successor of a node. This convention imitates English, which is written from left to right.

The following sections present the inquiry procedure for each of the three phases.

3.1 Failure in Phase One

Because network delays are bounded, if any of the users or MIXes produces no (signed) output, the consumer of that output and $\mathcal{O}_1$ will eventually detect the failure. The failing node is marked as *down* and excluded from the protocol.

[1] This restriction is removed in section 4.2

On the other hand, a node (MIX or user) can output an arbitrary (but signed) output which is not according to the format defined in Eqns. (1) and (2). Moreover, as s_n^l is based on public key cryptography and the K_n^l are random, any entity, in particular a MIX, can form a valid-looking onion that does not belong to any of the users in $\mathcal{U}$.

If user P_i is one of the observers in O_1 and the output S_d of MIX M_d does not contain s_i^{d+1} as it should, P_i will immediately detect the transgression. User P_i will publicly accuse M_d of cheating.

This means that M_d will have to prove its innocence by showing that it has properly decrypted all the onions contained in S_{d-1}. Let $\hat{s}_n^{d+1}$ denote the onions contained in purportedly erroneous S_d. Similarly, let $\hat{s}_n^d$ denote the onions contained in possibly erroneous S_{d-1}.

To prove proper decryption, M_d shall reveal all keys $\hat{K}_n^d$ and its permutation $\hat{\pi}_d$. Anyone can then verify whether $\hat{s}_{\hat{\pi}_d(i)}^{d+1} = \mathsf{E}_d\big(\hat{K}_i^d \| \hat{s}_i^d\big)$, for $i = 1, \ldots, N$. If all equalities hold for all N equations, then by the target collision-resistance assumption (see Section 1.1) any verifier is convinced of M_d's innocence.

Whenever a MIX is cleared, the onus of proof shifts to the preceding MIX. A MIX clears itself by revealing the keys in its possession (i.e. the decryption of its inputs) and its random permutation. The verifiers check that the decryptions match the MIX's inputs.

The check for the first MIX is only slightly different. MIX M_1 reveals the keys in its possession and its permutation $\hat{\pi}_1$. The verifiers proceed to check that all the signed inputs s_i^1 from all the users are such that the equality $s_i^1 = \mathsf{E}_1\big(\hat{K}_i^1 \| \hat{s}_i^1\big)$ holds for $i = 1, \ldots, N$.

If a MIX cannot prove its innocence, it is declared a disrupter. On the other hand, if M_d and all MIXes to the left of M_d can prove correct behavior, the plaintiff, in our example user P_i, is declared a slanderer.

Note that, if P_i is not one of the observers, any MIX can substitute an onion of its own making in place of the onion created by P_i. Thus, no-one will detect that a MIX cheated until the next phase (see next Section 3.2).

3.2 Failure in Phase Two

As in phase one, if a MIX fails to send a signed batch-commitment within the allowed time-frame, then the MIX is marked down and excluded from the protocol.

As mentioned in Section 2.2, $\mathcal{U} \subseteq \mathcal{O}_2$, meaning that each batch-commitment is received by all users. A user will complain and start an inquiry if it cannot find a commitment corresponding to one of its shared keys in the batch-commitment of a MIX. A user will also complain if it finds duplicate commitments corresponding to its key.

Let user P_i be the plaintiff, M_d be the defendant, and batch-commitment C_d be the object of litigation. An honest P_i will initiate an inquiry whenever it cannot find one and only one $\mathsf{com}\big[\delta_{d-1}(i) \| K_i^d\big]$ in C_d.

The defendant, M_d, will prove its innocence by revealing its permutation $\hat{\pi}_d$ and all the keys $\hat{K}_n^d$ in its possession. The verifiers check that the revealed permutation and keys agree with C_d. If the check fails M_d is declared a disrupter and excluded from the protocol. Otherwise, the verifiers proceed to check that the revealed keys agree with the preceding MIX's outputs of phase one. As in Section 3.1, this is done by verifying that the equality $\hat{s}_{\hat{\pi}_d(i)}^{d+1} = \mathsf{E}_d\big(\hat{K}_i^d \| \hat{s}_i^d\big)$ holds for all $i = 1, \ldots, N$.

If all equations hold, the defendant is innocent and the onus of proof shifts to the preceding MIX, namely M_{d-1}. If a MIX cannot prove its innocence, it is declared a disrupter. On the other hand, if all MIXes M_k, with $k \leq d$, can prove correct behavior, the plaintiff, in this case user P_i, is slandering and consequently labeled a disrupter.

3.3 Failure in Phase Three

Failure in the two preceding phases and the consequent inquiry do not endanger privacy. In phases one and two, the user has not yet sent a message to its correspondent. In phase three that is not the case. We must ensure that an inquiry does not betray an honest user's privacy.

As in the preceding two phases, if a MIX fails to produce a signed output within the allowed time frame, then the MIX is marked down and excluded from the protocol.

At the sending phase, a MIX can (actively) disrupt the protocol by several means.

1. It can output an erroneous message.
2. It can accept an erroneous message that has failed the integrity check.
3. It can initiate an unwarranted inquiry (see below).

Let MIX M_d be a disrupter such that instead of the output o_i^{d+1}, as defined in Eqs. (8) and (9) and belonging to user P_i, it outputs $\hat{o}_i^{d+1} \neq o_i^{d+1}$.

Proposition 1. *Given an honest user P_i, any modification of an onion belonging to P_i will be detected by the first honest MIX M_h to the right of the disrupter.*

Proof: By the integrity assumption (see Section 1.1), without knowing key K_i^h, it is impossible to create an encryption that passes the integrity check at M_h. The security of both the symmetric and asymmetric encryptions in addition to the hiding property of the commitment scheme (Section 1.1) ensure that only honest P_i and M_h know K_i^h. □

Whenever an onion fails the integrity check, the MIX at which the check failed initiates an inquiry. To avoid unwarranted investigations, MIX M_h can start an inquiry only if it can prove that it has performed the decryption with the correct key on the correct input but the result failed the integrity check. Adherence to the protocol implies that user P_i's onion be at the $\delta_{h-1}(i)^{\text{th}}$ position in O_{h-1} and at the $\delta_h(i)^{\text{th}}$ position in O_h. Moreover, during phase two, MIX M_h emitted $\mathsf{com}\left[\pi_h^{-1}(\delta_h(i))||K^h_{\rho_h(\delta_h(i))}\right] = \mathsf{com}\left[\delta_{h-1}(i)||K_i^h\right]$ at the $\delta_h(i)^{\text{th}}$ position in C_h.

To clear itself, MIX M_h will reveal $\delta_{h-1}(i)$, $\delta_h(i)$ and K_i^h. The verifiers will check that $\mathsf{com}\left[\delta_{h-1}(i)||K_i^h\right]$ is at the $\delta_h(i)^{\text{th}}$ position in C_h. If the checks fail, M_h is declared a disrupter for initiating an unsubstantiated inquiry.

Otherwise, the verifiers are convinced that key K_i^h should be applied on the $\delta_{h-1}(i)^{\text{th}}$ input (in O_{h-1}) and should result in the $\delta_h(i)^{\text{th}}$ output (in O_h).

Let $\hat{o}^{h-1}_{\delta_{h-1}(i)}$ denote the actual $\delta_{h-1}(i)^{\text{th}}$ message in O_{h-1}. The verifiers proceed to check that decryption of $\hat{o}^{h-1}_{\delta_{h-1}(i)}$ under key K_i^h fails the integrity check. If the integrity check succeeds, this means that M_h had no reason to interrupt the protocol. MIX M_h is declared a disrupter for initiating an unwarranted inquiry. Otherwise, if the integrity check fails, M_h is honest and correctly detected misbehavior on the part of a preceding MIX. The onus of proof is shifted to the preceding MIX M_{h-1}.

Each MIX is asked to clear itself until a dishonest MIX has been found. A suspect MIX M_l will clear itself by revealing $\delta_{l-1}(i)$ and K_i^l. Note that $\delta_l(i)$ was fixed when the successor MIX M_{l+1} cleared itself. The remaining inquiry steps remain unchanged.

The first MIX M_1 proves its innocence by revealing i and K_i^1. Note that disclosing i reveals the identity of P_i. The verifiers perform the same checks on M_1 as on the other MIXes.

If M_1 can innocent itself, then it must be the case that P_i intentionally created an onion that failed the integrity check at M_h. It follows that P_i is a disrupter. Thus, the protocol will have only revealed the identity of a corrupt user.

Note that all outputs are signed. Hence, a node cannot later deny having sent out an output that incriminates it.

Given the above protocol definition, we have

Proposition 2. *The protocol will rapidly designate the MIX that initiated an unwarranted inquiry as a disrupter.*

Proof: A dishonest M_d, with $d > 1$, will wrongly initiate an inquiry by pretending that an input from the preceding node fails the integrity check. Let the purportedly erroneous onion be at the j^{th} position in O_{d-1}. The j^{th} output in O_{d-1} belongs to user P_u, where $u = \rho_{d-1}(j)$. Using this notation, the onion that is the subject of the litigation is written o_u^d and the key shared between P_u and M_d is written K_u^d.

Let x be the correct decryption of o_u^d under key K_u^d. By the target collision-resistance assumption, M_d cannot find $x' \neq x$ such that $e_{K_u^d}(x') = o_u^d$. But x (being the correct decryption) will pass the integrity check.

The only remaining possibility is for M_d to reveal a wrongly shared key, $\hat{K}_u^d \neq K_u^d$.

However, M_d committed to the string $j||K_u^d$ in phase two. By the binding property of the commitment scheme, M_d cannot open for any value other than $j||K_u^d$. Moreover, M_d could not have committed to a different string because of the check performed by P_u at the end of phase two. By the non-interactive verifiability property, user P_u can perform this check independently of M_d because P_u knows both K_u^d (having generated it) and j (having studied $C_1, \ldots C_{d-1}$).

The argument for a dishonest first MIX, i.e. $d = 1$, is quite similar. □

Proposition 3. *A inquiry will always stop at the rightmost disrupter that which did not operate according to its commitment of phase two. The MIX initiating the inquiry reveals the position of the output corresponding to the erroneous input.*

Proof: The wording of the proposition implies the existence of an inquiry. Let M_h be the MIX initiating the inquiry. If the inquiry is unjustified, M_h will be designated as a disrupter (Proposition 2).

Let us now assume that the inquiry is justified. Let M_d, with $d > 1$, be the disrupter and P_i the user whose onion was altered. That is, M_d received o_i^{d-1} at the $\delta_{d-1}(i)^{\text{th}}$ position in O_{d-1} and produced $\hat{o}_i^{d-1} \neq o_i^{d-1}$ at the $\delta_d(i)^{\text{th}}$ position in O_d. Except for notational differences, the following reasoning also holds for the case $d = 1$. We will not treat this special case separately.

Without loss of generality, let M_h, where $h = d + k$, with $k \geq 1$, be the MIX initiating the inquiry. There are two possibilities:

1. MIXes $M_{d+1}, \ldots M_{d+k-1}$ ignored the failed integrity checks on the erroneous onion.
2. M_d got hold of the keys $K_i^{d+1}, \ldots, K_i^{d+k-1}$ and formed an onion that passed the integrity check performed by its successors $M_{d+1}, \ldots M_{d+k-1}$, but not the check at M_h.

In the first case, M_{d+k-1}, i.e M_{h-1}, will quickly be incriminated. Indeed, there is no way M_{h-1} can justify its output given the input from its predecessor. Either M_{h-1} reveals the wrong key and is incriminated (by the binding property), or reveals the correct key, disclosing to all that the input from the predecessor never passed the integrity check.

In the second case, M_d will be incriminated. The successors of M_d can all justify their output. That is not the case for M_d. Either M_d opens for the wrong key and is incriminated, or it opens for the right key in which case M_d's input and output do not match.

In case all MIXes to the left of M_h can clear themselves, then P_i must have intentionally created a faulty onion. The protocol would have revealed the rightmost disrupter, in this case a user.

The second part of the proposition is true by construction. □

Let the last MIX M_L be honest and all other MIXes dishonest. In that case, the colluding MIXes can form a fake onion using keys that are compatible with the commitments contained in $C_1, \ldots, C_{L-1}$. Seeing the fake onion, the honest and last MIX M_L will initiate inquiry because the onion cannot possibly pass the integrity check (Proposition 1). Interestingly enough, all MIXes but the first can prove their innocence. The first MIX cannot because it cannot link its fake output to a signed input from a user and simultaneously pass the commitment check (Proposition 3).

Note that the dishonest MIXes have managed to force the honest MIX M_L to reveal the position of the output corresponding to the erroneous output. In other words, $L-1$ dishonest MIXes can collude to successfully trace messages. Untraceability has been lost: if there is only one honest MIX, privacy cannot be guaranteed.

Proposition 4. *Two honest MIXes M_i and M_j, with $i < j$, guarantee that any disrupter to the left of M_j will be identified and still protect the privacy of all honest users.*

Proof: By Proposition 1 and 3, any disrupting MIX to the left of M_i will be identified. Although MIX M_i will reveal the output position corresponding to the erroneous input, M_j, which was not involved in the inquiry, still protects all users' privacy.

Disruptions cannot occur both to the left and to the right of any honest MIX, in particular M_i, because an honest MIX will detect any disruption on the left and demand an inquiry. This, before the disruption on the right can take place.

Any disruption by colluding MIXes sandwiched between M_i and M_j will be detected by M_j. The ensuing inquiry will designate the rightmost disrupter that did not operate according to its commitment of phase two. By Proposition 3, M_i is not involved in the inquiry and will continue to protect user's privacy. □

Proposition 4 assumes honest users because dishonest users can always manage to pass through M_i and M_j but still fail at a later MIX.

Remark: In phases one and two, each defendant had to reveal its complete permutation function and all keys in its possession. In phase three, only a specific permutation value and key (corresponding to the erroneous onion) have to be revealed. Thus, contrary to phases one and two, the inquiry of phase three does not affect all inputs, only the faulty ones.

4 Extending the Protocol

In this section we will extend the protocol defined in the preceding two sections to achieve fault-removal capability (preserving privacy) but assuming only one trusted MIX.

In Section 4.1 we add the capability to detect garbage output from the last MIX. In Section 4.2 we show how the requirement, that at least two MIXes are honest, can be relaxed to assume only one honest MIX.

4.1 Final MIX Problem

Our protocol as described so far does not impose any restriction on the format of the messages sent by users to their correspondent. An interesting problem occurs when the last MIX (or more generally the rightmost

MIXes) is dishonest and simply decides to output garbage. Users may detect that the last MIX is dishonest but cannot accuse M_L lest they lose their untraceability.

On the other hand, if the messages have an intrinsic validity check, like in voting schemes or some payment systems, then a dishonest last MIX will be prosecuted. Any observer of M_L's invalid output will detect wrongdoing and demand an inquiry.

Our protocol can be extended to impose an integrity check on the final output by the last MIX. To achieve this, each user P_n will randomly generate a public/private key pair in phase one. Let α_n designate this random secret key and β_n the public key. User P_n will add β_n to the onion o_n^L defined in Eq. (1). At the end of phase one, the last MIX will output each of these random public keys according to δ_L. Thus, β_n will be output at the $\delta_L(n)^{\text{th}}$ position. The users reserve the right to complain if their public key was not output properly by M_L.

At the sending phase, user P_n will sign $\mathcal{A}_n \| m_n$ in Eq. (8), using private key α_n, before the encryption with key K_n^L. MIX M_L will output the α_n signed $\mathcal{A}_n \| m_n$. Using public key β_n, any observer of the protocol can check whether the output at the $\delta_L(n)^{\text{th}}$ position was properly formed and demand an inquiry if need be.

Henceforth, we shall refer to this extended protocol as protocol E1. This extension is inspired by the digital pseudonyms as described in [2] and nicely re-explained in Section 2.3 of [10].

Proposition 5. *In protocol E1, two honest MIXes M_i and M_j, with $i < j$, guarantee that any disrupter will be identified and still protect all honest users privacy.*

Proof: The results of Proposition 4 hold in the extended protocol. If the last MIX is honest ($j = L$), Proposition 5 and 4 are equivalent.

To prove Proposition 5 we need to demonstrate that disrupters to the right of M_j, with $j < L$, will also be detected without violating the privacy of honest users.

Assuming an honest user, the signature verification on the final message ensures that only the correct user could have created the final output. If this check fails, then there must be a disrupter to the right of M_j. By Proposition 3, the ensuing inquiry will designate the rightmost disrupter M_d, with $d > j$, that is unable to justify its output. As $d > j$, honest MIX M_j is not involved in the inquiry and continues to protect users privacy. □

Only a dishonest originator can fool all integrity checks performed by all MIXes and still cause the validity check performed on the output of the last MIX to fail. An inquiry will reveal the identity of the disrupting user, which is of course acceptable.

4.2 Single Trusted MIX

Let $I = 2L$ represent the number of MIXes in the cascade. We pair the MIXes two by two. Let p_l denote the pair (M_{2l-1}, M_{2l}), with $l = 1, \ldots, L$. Suppose that each pair of MIXes p_l is controlled by a single entity. Let us call an entity controlling such a pair of MIXes a 2MIX.

Proposition 6. *In protocol E1, an honest 2MIX guarantees that any disrupter will be identified and still protect all honest users' privacy.*

Proof: Proposition 5 applies to any two distinct MIXes M_i and M_j, in particular if $j = i + 1$ with i odd. □

We use "left part" to refer to the first and "right part" to refer to the second MIX in a 2MIX.

As both the left and the right part are controlled by the same entity, we can let the two parts have the same public key. We can save two exponentiations (a signature and a verification) if the outputs of the left and the right part are concatenated and signed together.

From the users perspective, a 2MIX is merely two adjacent MIXes sharing a private key. The onions that user P_n computes in phase one are modified to contain two keys K_{2l-1}^n, K_{2l}^n, one for the left and one for the right MIX of 2MIX number l. Similarly, the onions that user P_n computes in phase three are modified to have two encryptions, once with key K_{2l}^n and once with key K_{2l-1}^n.

The optimized version of the 2MIX extension of the E1 protocol is called the E2 protocol.

5 Efficiency

In this section we analyze the computational cost of the E2 protocol.

Let E, D, S, V, e, d, c denote, respectively, the cost of computing a public key encryption, a public key decryption, a signature, verifying a signature, symmetric key encryption, symmetric key decryption, and a commitment using the com function. Let L represent the number of 2MIXes.

In phase one, each user computes L public key encryptions, one for each 2MIX, and signs the resulting onion. In phase two, each user has to verify the signature on the output of each 2MIX and compute an commitment to verify the commitment on the left and right parts of each 2MIX. It then has to sign a message showing its agreement with the results of phase two. In phase three, each user P_n will sign its final output of the cascade using random secret key α_n. Additionally, each 2MIX costs two shared key encryptions, and the resulting input to the cascade must be signed with the users long-term secret key. These computational costs per user are summarized in Table 2.

Table 2: Computational cost for each user

Phase one	$LE + S$
Phase two	$LV + 2Lc + S$
Phase three	$2S + 2Le$
Total cost	$L(E + V + 2e + 2c) + 4S$

In phase one, each 2MIX – other than the first – verifies its predecessor's output, decrypts N public key encryptions (once for each user), and signs its output. The first 2MIX has the additional cost of verifying the signature on the output of each user. In phase two, each 2MIX has to compute two commitments per user and sign its output. In phase three, each 2MIX – other than the first – verifies its predecessor's output, performs two shared key decryptions per user, and signs its output. The first 2MIX has the additional task of verifying the signature of each user. The computational costs per 2MIX are summarized in Table 3.

Table 3: Cost at each 2MIX (other than the first)

Phase one	$V + ND + S$
Phase two	$S + 2Nc$
Phase three	$V + 2Nd + S$
Total cost	$3S + 2V + N(D + 2c + 2d)$

As mentioned earlier, phase three can be repeated to send multiple messages without having to go through phases one and two. The cost of the third phase will become predominant as the number of messages sent increases.

The protocol described in [1] requires $4\sigma + 5.2N + 2.2$ exponentiations per MIX, where $\sigma = -\log\varepsilon$ and ε denotes the maximum success probability of an attack.

Given a quorum size k, the MIX cascade defined in [4] requires approximately $3\kappa kN + 7k^2N$ exponentiations for all MIXes, where $\kappa = 1 - \log\varepsilon/2\log N$.

Let $\varepsilon = 2^{-80}$, $N = 2^{10}$, $k = L$, and $L = 5$. Inserting these values, we obtain about 325 exponentiations per user per MIX for [1] and 50 exponentiations per user per MIX for [4]. Protocol E2 with L 2MIXes would necessitate about 1.4 exponentiations and about 5 non-public key computations per user per MIX (averaged over L 2MIXes), for all three phases combined. Note also that exponentiation is about three orders of magnitude more expensive than non-public key operations such as hash functions or shared key encryption/decryption.

6 Conclusion

We have described a fault-removing MIX cascade guaranteeing privacy of senders as long as a single MIX can be assumed to be honest. The MIX cascade will unambiguously diagnose faulty nodes, even malicious ones, and remove them from the protocol.

The MIX cascade is based on rather basic cryptographic primitives. We do not make use of more sophisticated interactive zero-knowledge proofs. Our protocol has three phases, a key-sharing phase, a commitment phase, and a sending phase. Existing non-dependable systems usually consist of two phases, a key-sharing phase and a sending phase. Moreover, they do not impose an integrity check to be appended into each encryption. We achieve fault-removal capability at the expense of an integrity check and of the additional commitment phase.

Once the first two phases are completed, the sending phase can be repeated to send multiple messages. As a result, our cascade can transmit untraceable messages of any length at low computational cost.

Compared to other dependable MIX cascades, our cascade is significantly more efficient as well as simpler, albeit at the cost of error recovery.

By relaxing the one honest MIX hypothesis and resorting to threshold cryptography, one could possibly circumvent faulty MIXes—hence allow error recovery—but still guarantee privacy. How this may actually be done is the subject of future work.

References

[1] ABE, M. Universally verifiable mix-net with verification work independent of the number of mix-servers. In Nyberg [8], pp. 437–447.

[2] CHAUM, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *CACM 24*, 2 (Feb. 1981), 84–88.

[3] GÜLCÜ, C., AND TSUDIK, G. Mixing e-mail with babel. In *Proc. 1996 Symposium on Network and Distributed System Security* (Feb. 1996), pp. 2–16.

[4] JACOBSSON, M. A practical mix. In Nyberg [8], pp. 448–461.

[5] KELSEY, J., SCHNEIER, B., AND WAGNER, D. Protocol interactions and the chosen protocol attack. *Lecture Notes in Computer Science 1361* (1998), 91–104.

[6] LAPRIE, J. C., Ed. *Dependability: Basic Concepts and Terminology.* Springer-Verlag, 1992.

[7] MENEZES, A. J., VAN OORSCHOT, P. C., AND VANSTONE, S. A. *Handbook of Applied Cryptography.* CRC Press, 1997.

[8] NYBERG, K., Ed. *Advances in Cryptology—EUROCRYPT 98* (May 1998), vol. 925 of *Lecture Notes in Computer Science*, Springer-Verlag, 1998.

[9] OGATA, W., KUROSAWA, K., SAKO, K., AND TAKATANI, K. Fault tolerant anonymous channel. *Lecture Notes in Computer Science 1334* (1997), 440–444.

[10] PARK, C., ITOH, K., AND KUROSAWA, K. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology—EUROCRYPT 93* (23–27 May 1993), T. Helleseth, Ed., vol. 765 of *Lecture Notes in Computer Science*, Springer-Verlag, 1994, pp. 248–259.

[11] PFITZMANN, A., PFITZMANN, B., AND WAIDNER, M. Isdn-mixes: Untraceable communication with very small bandwidth overhead. In *GI/ITG Conference: Communication in Distributed Systems* (February 1991), Springer-Verlag, Heildelberg 1991, pp. 451–463.

[12] SAKO, K., AND KILIAN, J. Receipt-free mix-type voting scheme: A practical solution to the implementation of a voting booth. In *Advances in Cryptology—EUROCRYPT 95* (9–12 May 1995), L. Guillou and J.-J. Quisquater, Eds., vol. 921 of *Lecture Notes in Computer Science*, Springer-Verlag, 1995, pp. 393–403.

[13] STINSON, D. R. *Cryptography - Theory and Practice.* CRC Press, 1996.

[14] SYVERSON, P., GOLDSCHLAG, D., AND REED, M. Anonymous connections and onion routing. In *Proceedings of 18th Annual Symposium on the Symposium on Security and Privacy* (May 1997), IEEE CS Press, pp. 44–54.