How to Break, Fix, and Optimize “Optimistic Mix For Exit-Polls”

Douglas Wikström, douglas@sics.se
Swedish Institute of Computer Science (SICS)
Box 1263, S-164 29 Kista, Sweden

December 6, 2002

Abstract

First we present two attacks for the mix-net proposed by Golle et al. [11], and also propose modifications that counter our attacks. The first attack breaks the privacy of the protocol completely.

Our attacks are adaptations of the “relation attack”, discussed by Jakobsson [14], Pfitzmann [29, 28], and Wikström [31], but we introduce a novel way of exploiting intermediate values of different mix-sessions.

Then we propose two optimizations of the protocol that reduce the number of exponentiations computed by each mix-server from $4(k + 1)N$ to $4N$, where $k$ is the number of mix-servers, and $N$ is the (large) number of senders.

Thus the modified protocol outperforms the original by a factor $k + 1$, with complexity essentially independent of $k$.

Keywords: mix-net, anonymous channel, electronic voting.

---

1We improve the analysis of the original protocol from $(5 + 10k)N$ to $4(k + 1)N$. 
How to Break, Fix, and Optimize “Optimistic Mix For Exit-Polls”

Douglas Wikström, douglas@sics.se
Swedish Institute of Computer Science (SICS)
Box 1263, S-164 29 Kista, Sweden

December 6, 2002

Abstract

First we present two attacks for the mix-net proposed by Golle et al. [11], and also propose modifications that counter our attacks. The first attack breaks the privacy of the protocol completely.

Our attacks are adaptations of the “relation attack”, discussed by Jakobsson [14], Pfitzmann [29, 28], and Wikström [31], but we introduce a novel way of exploiting intermediate values of different mix-sessions.

Then we propose two optimizations of the protocol that reduce the number of exponentiations computed by each mix-server from \(4(k+1)N\) to \(4N\), where \(k\) is the number of mix-servers, and \(N\) is the (large) number of senders.

Thus the modified protocol outperforms the original by a factor \(k+1\), with complexity essentially independent of \(k\).

1 Introduction

The notion of a mix-net was invented by Chaum [3], and further developed by a number of people. Properly constructed a mix-net enables a set of senders to send messages anonymously. A mix-net can be viewed as an electronic analog of a tombola; messages are put into envelopes, the envelopes are mixed, and finally opened. It is impossible to tell who sent any given message. Thus the service that a mix-net provides is privacy.

Informally the requirements on a mix-net are: correctness, privacy, robustness, availability, and efficiency. Correctness implies that the result is correct given that all mix-servers are honest. Privacy implies that if a fixed minimum number of mix-servers are honest privacy of the sender of a message is ensured. Robustness implies that if a fixed number of mix-servers are honest, then any attempt to cheat is detected and defeated. Availability and efficiency are the general requirements on any system run on an open network.

A mix-net consists of a number of mix-servers, i.e. servers, that collectively execute a protocol. The basic idea of a mix-net, present already in Chaum’s work [3], is that each mix-server receives a list of encrypted messages, transforms them, using partial decryption or random re-encryption, reorders them,

\(^{1}\)We improve the analysis of the original protocol from \((5 + 10k)N\) to \(4(k+1)N\).
and then outputs the transformed and reordered list. It should be difficult to find an element in the input list, and an element in the output list that encrypts the same message. The reason for using several independent mix-servers is that it allows a sender to trust a subset of the mix-servers to ensure privacy. Later constructions have mostly dealt with robustness, availability and efficiency, which are aspects ignored by Chaum.

1.1 Previous Work and Applications of Mix-Nets

The mixing paradigm has been used to accomplish privacy in many different scenarios. Chaum’s original “anonymous channel” [3, 24] enables a sender to securely send mail to a receiver anonymously, and also to securely receive mail from this receiver without revealing the sender’s identity. When constructing election schemes [3, 8, 26, 30, 23] the mix-net is used to ensure that the vote of a given voter can not be revealed. Also in the construction of electronic cash systems [15] mix-nets have been used to ensure privacy. Thus a mix-net is a useful primitive in constructing cryptographic protocols.

Abe gives an efficient construction of a general mix-net [1], and argues about its properties. Jakobsson has written (partly with Juels) a number of more general papers on the topic of mixing [14, 16, 17] also focusing on efficiency, of which the first appeared at the same time as Abe’s construction.

There has been a breakthrough in the construction of zero-knowledge proofs of a correct shuffle recently. Furukawa and Sako [9], and Neff [22] respectively have both found efficient ways to compute such proofs.

A promising approach to practical mixing is given by Golle et al. [11]. They combine a beautiful robustness test with roots in the work by Jakobsson and Juels, with the notion of “double enveloping”. The latter notion is introduced independently by Wikström [32], except that he uses different keys for the two layers.


2 Outline of the Paper

For readers not familiar with the protocol of Golle et al. [11], we present a short review of this protocol in Appendix A.

We present two attacks for this mix-net, of which the first breaks the privacy of the protocol completely. This is followed by an informal discussion on the issue of concurrent mix-sessions. Then we describe modifications to the original protocol that counter our attacks.

Before we present our optimizations we review and discuss the original method of joint decryption, and improve the analysis of the original protocol. Finally we describe the modifications to the protocol that reduce the number of exponentiations computed by each mix-server. The modifications we propose are adaptations of earlier work by Wikström [32].
3 The Attacks

The goal of the adversary, who we in this paper call Eve, is to break the privacy of our typical honest sender Alice.

The attacks are adaptations of the “relation attack”, discussed by Jakobsson [14], Pfitzmann [29, 28], and later in a slightly more general setting by Wikström [31], to the setting with double enveloping. The idea of the “relation attack” is that to break the privacy of Alice, Eve computes a cryptotext of a message related to Alice’s message. Then the mix-net is run as usual. The output of the mix-net contains two messages related in a way chosen by Eve. Some types of relations let Eve determine the message sent by Alice. To avoid the “relation attack” the mix-net must be CCA-secure in some sense.

We consider only two attacks, but there are some natural variants of these.

3.1 Sending a Message to the Mix-Net

We first review what a typical honest sender Alice computes in the original protocol to send a message $m$ to the mix-net. Let $y$ be the public key of an ElGamal [7] cryptosystem in a group $G_Q$, of prime order $Q$, generated by $g$. Alice computes:

$$(u, v) = E_y(m), \quad w = h(u, v), \quad \alpha = (E_y(u), E_y(v), E_y(w)) = ((\mu_1, \mu_2), (v_1, v_2), (\omega_1, \omega_2)),$$

where $h$ is a hash function modeled by a random oracle. Then Alice computes a zero-knowledge proof of knowledge $\pi_{id}(u, v, w)$, that depends on the current mix-session identifier $id$. Finally Alice sends $(\alpha, \pi_{id}(u, v, w))$ to the “bulletin board”.

3.2 Attack for the Original Protocol with Honest Mix-Servers

Alice is our typical sender, and we show that in the original protocol of Golle et al. [11], Eve can break her privacy. Eve does the following:

1. Eve chooses $\delta$ and $\gamma$ randomly in $\mathbb{Z}_Q$, and computes:

$$w_\delta = h(\mu_1^\delta, \mu_2^\delta), \quad \alpha_\delta = (E_y(\mu_1^\delta), E_y(\mu_2^\delta), E_y(w_\delta)),$$

and

$$w_\gamma = h(\mu_1^\gamma, \mu_2^\gamma), \quad \alpha_\gamma = (E_y(\mu_1^\gamma), E_y(\mu_2^\gamma), E_y(w_\gamma)).$$

Then Eve computes corresponding proofs of knowledge $\pi_{id}(\mu_1^\delta, \mu_2^\delta, w_\delta)$ and $\pi_{id}(\mu_1^\gamma, \mu_2^\gamma, w_\gamma)$, of $(\mu_1^\delta, \mu_2^\delta, w_\delta)$ and $(\mu_1^\gamma, \mu_2^\gamma, w_\gamma)$ respectively. This gives Eve two perfectly valid triples with corresponding proofs, that she sends to the “bulletin board” (possibly by corrupting two senders).

2. Eve waits until the mix-net has successfully completed its execution. During the execution of the mix-net, the mixes first jointly decrypt the “outer layer” of the double encrypted messages. This results in a list of triples of the form $(u, v, h(u, v))$, where the benign triples have already been deleted. Eve forms a list by choosing the first two components $(u, v)$, i.e. the inner El Gamal pair, from each triple:

$$((u_1, v_1), \ldots, (u_N, v_N)) \quad (1)$$
The final output of the mix-net is a list of cleartext messages denoted 
\((m_1, \ldots, m_N)\).

3. Eve computes \(\delta \gamma^{-1} \mod Q\), which is easy since \(Q\) is prime, and the list 
\((m'_1, \ldots, m'_N) = (m_1^{\delta/\gamma}, \ldots, m_N^{\delta/\gamma})\),
and then finds a pair \((m_i, m'_j)\) such that \(m_i = m'_j\). From this she concludes that with very high probability \(m_j = u^\gamma\). Then she computes \(z = m_1^{1/\gamma}\) and finds a pair \((u_l, v_l)\) in the list (1) such that \(z = u_l\). Finally she concludes that with very high probability \(m_l\) was the message sent by Alice to the mix-net.

3.2.1 Why the Attack is Possible.

The attack exploits two different flaws of the protocol. The first is that the sender of a message, e.g. Alice, proves only knowledge of the inner El Gamal pair \((u, v)\) and the hash \(w = h(u, v)\), and not knowledge of the message \(m\). This allows Eve to compute a single encrypted message \((\mu_1^{\delta}, \mu_2^{\delta})\) of a power \(u^{\delta}\) of the first component \(u\) of the inner El Gamal pair of the double encrypted message sent by Alice, and a perfectly valid proof of knowledge of \((\mu_1^{\delta}, \mu_2^{\delta})\) and \(w_3 = h(\mu_1^{\delta}, \mu_2^{\delta})\) (and similarly for \((\mu_1^{\gamma}, \mu_2^{\gamma})\)).

The second flaw is that identical El Gamal keys are used for both the inner and outer El Gamal system. This allows Eve to use the mix-net to decrypt \((\mu_1^{\gamma}, \mu_2^{\gamma})\), and thus get her hands on \(u^\gamma\) (and similarly for \(u^\delta\)). Eve can then identify \((u, v)\) as the inner El Gamal pair of Alice, which breaks her privacy.

remark 1 Eve does not have to send 2K messages (corrupt 2K senders in the second mix-session) to break the privacy of K honest senders. At additional computational cost a smaller number of messages suffice. Suppose Eve wants to break the privacy also of Alice’s sister who sent a message \(m'\) encrypted as follows:

\[
\begin{align*}
(u', v') &= E_y(m'), \\
\alpha' &= (E_y(u'), E_y(v'), E_y(w')) = ((\mu_1^{\delta}, \mu_2^{\delta}), (v_1^{\gamma}, v_2^{\gamma}), (w_1^{\gamma}, w_2^{\gamma})).
\end{align*}
\]

Then Eve performs the attack above with the change that she starts with a single pair \((\mu_1^{\delta}, \mu_2^{\delta})\) for some randomly chosen \(\zeta\) instead of the two distinct pairs \((u_1, v_1)\) and \((u_1', v_1')\) that would have given two “unrelated” attacks. The only necessary change is that in the final step she computes the matrix \((\omega_i^l u_j)\) of all pairwise products of the first components in the list (1), and then identifies the pair \((l, l')\) such that \(u_l^l u_{l'}^l = z\). Finally she concludes that with high probability Alice sent \(m_l\), and Alice’s sister sent \(m_l'\). The approach is generalized to higher dimensions in the natural way, to break the privacy of several senders.

3.3 Attack for Different Keys and Corrupt Mix-Server

Suppose we change the protocol slightly by requiring that the mix-servers generate separate keys for the outer and inner El Gamal systems, to avoid the first attack of Section 3.2. That is, we assume that there are two different public
keys \( y_{in} = g^{x_{in}} \) and \( y_{out} = g^{x_{out}} \), with corresponding private keys \( x_{in} \) and \( x_{out} \), for the inner and outer system respectively, but employed in the same group \( G_Q \). This is the type of double enveloping proposed by Wikström [32].

For the following attack to work we need some additional assumptions.

### 3.3.1 Some Unclear Details of the Paper.

We start by quoting Section 5, under “Setup.” point 4 of Golle et al. [11], presenting the proof of knowledge \( \pi_{id}(u, v, w) \) of the sender Alice:

4. This proof of knowledge should be bound to a unique mix-session identifier to achieve security over multiple invocations of the mix.

Any user who fails to give the proof is disqualified, and the corresponding input is discarded.

If different keys are used for each mix-session then the above makes no sense, since the proof of knowledge of \( u, v \) and \( w \) already depends on the public key of the outer El Gamal system. There is clearly practical value in not changing keys between sessions.

From now on we assume that the keys are not changed between sessions, not even if a mix-server is found to be cheating. If a mix-server is found to be cheating its shared keys are instead reconstructed by the remaining mix-servers, and in later mix-sessions the actions of the cheating mix-server is performed either in the open or by the first remaining mix-server (the details of this does not matter).

The original paper of Golle et al. [11] does not explicitly say if the discovery of the corrupted mix-server results in a new execution of the key generation protocol. Apparently the intention of the authors is to let the remaining mix-servers generate a new set of keys if any cheating is discovered [12].

However, if a new key generation protocol is not run upon discovery of a cheating mix-server, we can give an attack on the protocol. We think the attack is interesting even though this interpretation is not the one intended by the authors, since it shows the importance of explicitly defining all details of the protocol. It also highlights some issues with running several mix-sessions using the same set of keys concurrently.

### 3.3.2 The Attack.

The attack is employed during two mix-sessions using the same keys. In the following we describe the actions of Eve during the first and second mix-sessions respectively.

**The First Mix-Session.** We assume that Alice and Alice’s sister have both sent inputs to the mix-net (and use the notation of Remark 1 for the input of Alice’s sister).

Suppose that Eve is able to corrupt the first mix-server in the mix-chain. The first mix-server then does the following. It replaces \( \alpha \) and \( \alpha' \) with:

\[
(E_{y_{out}}(u), E_{y_{out}}(v), E_{y_{out}}(w')), \text{ and } (E_{y_{out}}(u'), E_{y_{out}}(v'), E_{y_{out}}(w))
\]

respectively, in its input list, i.e. the third components of the two triples are shifted. Then it takes the altered input list and simulates a completely honest
mix-server on this input. Note that the product test can easily be fulfilled by
the cheating mix-server.

At the end of the mixing the mix-servers verify the tuples, and discover the
invalid tuples \((u, v, w)\) and \((u', v', w)\). These tuples are traced back all the way
to the first mix-server, which is revealed as a cheater. In this process Eve is able
to link Alice to \((u, v)\) (and Alice’s sister to \((u', v')\)).

Finally the honest mix-servers finish the protocol by using the general con-
structions based on the work by Neff [22] as in Golle et al. [11].

remark 2 Note that the method can easily be generalized to several senders, by
using a circular shift of the third components. Other variants of this step are
also possible, i.e. all that is needed is that Alice is linked to her inner El Gamal
pair \((u, v)\).

The Second Mix-Session. To allow the mix-net to execute a second mix-
session using the same set of keys, the cheater’s key is recreated by a quorum of
the mix-servers, so that they can emulate the actions of the cheating mix-servers
in the next mix-session.

At this point Eve knows that Alice sent \((u, v)\). To determine the contents
of this El Gamal pair, Eve waits for the next mix-session using the same set of
keys. Then she uses the second mix-session to decrypt \((u, v)\), and the “relation
attack” to identify the decrypted cleartext. That is, Eve does the following:

1. Eve chooses \(\delta\) and \(\gamma\) randomly in \(\mathbb{Z}_Q\), and computes:

\[
\begin{align*}
\alpha_{\delta} &= (E_{y_{\text{out}}}(u^\delta), E_{y_{\text{out}}}(v^\delta), E_{y_{\text{out}}}(w^\delta)) , \\
\alpha_{\gamma} &= (E_{y_{\text{out}}}(u^\gamma), E_{y_{\text{out}}}(v^\gamma), E_{y_{\text{out}}}(w^\gamma)) .
\end{align*}
\]

Then Eve computes corresponding proofs of knowledge of \((u^\delta, w^\delta)\) and
\((u^\gamma, w^\gamma)\) respectively. This gives Eve two perfectly valid triples with
corresponding proofs that she sends to the “bulletin board” (possibly by
corrupting two senders).

2. Eve waits until the mix-net has successfully completed its execution. The
final output of the mix-net is a list of cleartext messages denoted

\[
(m_1, \ldots, m_N) .
\]

3. Note that \(m_i = m^\delta\) and \(m_j = m^\gamma\) for some \(i\) and \(j\). Eve computes
\(\delta\gamma^{-1} \mod Q\), computes the list

\[
(m'_1, \ldots, m'_N) = (m_i^{\delta/\gamma}, \ldots, m_N^{\delta/\gamma}) ,
\]

and finally finds a pair \((m_i, m'_j)\) such that \(m_i = m'_j\). From this she
concludes that with very high probability \(m_i^{1/\gamma}\) is the message sent by
Alice to the mix-net in the \textit{first} mix-session.

3.3.3 Why the Attack is Possible.

The attack exploits only one of the flaws of the protocol, namely that the sender
of a message proves knowledge of the inner El Gamal pair \((u, v)\) and \(w\), and not
knowledge of \(m\). This allows Eve to use the second mix-session as a decryption
oracle, and get her hands on \(m\).


4 Concurrency of Mix-Sessions

A simple countermeasure to the second attack explained in Section 3.3, where the keys of the inner and outer system are different, is to stipulate that if a cheating mix-server is identified, new keys must be generated for the next mix-session. That is, if a cheating mix-server is identified, the general methods developed by Neff [22] are used, and then before the next mix-session is started the joint key generation protocol is run to create new keys.

The disadvantage of this countermeasure is that the mix-net can not be allowed to execute several mix-sessions using the same keys concurrently. The problem is that if one mix-session is still receiving messages while another mix-session discovers a cheating mix-server, the second attack of Section 3.3 can still be applied (under the assumptions of Section 3.3). Note that it does not solve the problem to run the general methods developed by Neff [22] on all mix-sessions using the same keys at this point.

A natural attempt at solving the concurrency problem is to let some mix-sessions abort execution in the event of cheating, but that is not an alternative since it breaks the robustness of the mix-net.

This problem of concurrency may seem academic, since in most election scenarios it is not very cumbersome to have different keys for each mix-session. However, in future applications it may be advantageous to be able to run several different concurrent, and robust, mix-sessions using the same keys.

In the next section we propose modifications of the protocol that counter the attacks we have found. The modifications also seem to solve the concurrency problem.

5 How to Counter the Attacks

As explained above our attacks exploit two different flaws of the protocol: the lack of an appropriate proof of knowledge, and the use of identical keys for the inner and outer system.

The first and most powerful attack can be countered simply by using different keys for the inner and outer systems, as in Wikström [32]. This has also some additional advantages discussed in Section 7. However, from Section 4 above we know that using different keys does not suffice to give a good solution of the concurrency problem.

Another method of countering the first attack is to let the sender prove knowledge of the actual message sent. Intuitively this is a very natural requirement. The new proof of knowledge is described below.

Together these two changes of the protocol counter both our attacks, and seem to allow several and concurrent mix-sessions using the same set of keys.

5.1 How the Sender Proves Knowledge of $m$

The sender must prove knowledge of $m$, and all randomness used in the encryption process. In addition to this the proof should be bound to the mix-session identifier. If we choose the keys of the cryptosystem appropriately this turns out to be easy to accomplish.
Suppose that we let $g$ be the canonical generator of $G_Q$ for the outer El Gamal system, i.e. to generate a key pair for the outer system we pick $x_{out} \in \mathbb{Z}_Q$ randomly, compute $y_{out} = g^{x_{out}}$, and let $(g, y_{out})$ be the public key and $x_{out}$ the private key.

To generate keys for the inner system we pick $x_{in} \in \mathbb{Z}_Q$ randomly, compute $y_{in} = y_{out}^{x_{in}}$, and let $(y_{out}, y_{in})$ be the public key, and $x_{in}$ be the private key. The trick to use the public key $y_{out}$ to play the role of the canonical generator for the inner El Gamal system was introduced in Wikström [32].

Both keys are of course really generated during a joint key generation protocol, but that presents no additional problems.

Consider now the form of a double encrypted message $m$ sent to the bulletin board. It has the form:

$$[(g^{r_1}, y_{out}^{r_1}, y_{out}^{s_1}), (g^{r_2}, y_{out}^{r_2}(y_{in}^{s_1}m)), (g^{r_3}, y_{out}^{r_3}h(y_{out}^{s_1}, y_{in}^{s_1}m))] .$$

If we view $(g^{r_1}, r_1), (y_{out}^{r_1+s_1}, r_1 + s_1), (g^{r_2}, r_2), (g^{r_3}, r_3)$ as key pairs of Schnorr signature schemes, of which one uses the canonical generator $y_{out}$ instead of $g$ we simply sign the following information:

$$[(g^{r_1}, y_{out}^{r_1}, y_{out}^{s_1}), (g^{r_2}, y_{out}^{r_2}(y_{in}^{s_1}m)), (g^{r_3}, y_{out}^{r_3}h(y_{out}^{s_1}, y_{in}^{s_1}m))], \text{id} ,$$

using each of the three private keys. Thus the sender sends almost the same information using this solution. The only difference is that she proves not knowledge of $u = g^s$, and $v = y_{in}^m$, but instead knowledge of $m$, since the knowledge of $r_1$, $r_2$ and $r_1 + s$ determines $m$. The proof of knowledge is bound to a specific mix-session identifier similarly as sketched in the original paper.

This provides a type of CCA-security, since it seems that no sender can gain any knowledge that she did not already possess without corrupting at least one mix-server. We emphasize that this is not true for corrupted mix-servers, that exploit the fact that other mix-servers may perform some computations before they have discovered the cheating.

**Remark 3** Note that it is vital that the sender proves knowledge of all randomness used in the encryption process. Otherwise it may be possible to use some variant of the “relation attack” to break the privacy of Eve.

### 5.2 Summary of Modifications to Counter the Attacks

To summarize, we propose that:

1. The sender proves knowledge of all randomness used in the encryption of the message $m$ thereby proving knowledge of the message $m$, and the inner triple $(u, v, w)$, instead of just knowledge of the latter triple.

2. The keys of the inner and outer systems are generated independently, instead of using identical keys for the inner and outer layer.

### 6 Original Decryption Method and Improved Analysis

In the original protocol the authors point out that if there are $k$ mix-servers and they use a $(k, t)$ VSS-scheme they may decrypt the output of an El Gamal
ciphertext \((u, v) = (g^r, y^m)\), without explicitly reconstructing the El Gamal key. We assume here that we have a single shared El Gamal key pair \((x, y)\), where \(y = g^x\), for which the share of the \(i\):th server of the private key \(x\) is \(x_j\). A quorum \(T\) of \(t\) servers can then decrypt a ciphertext \((u, v)\) as follows:

\[D_x(u, v) = vu^{-x} = v \prod_{j \in T} (u^{x_j}) \prod_{l \neq j \in T} -\left(u^{x_j} \right)^{-1} \]

using Lagrange interpolation. This requires each mix-server to compute \(f_j = u^{x_j}\). To ensure that the mix-servers do indeed use their secret key \(x_j\) and not some other value, each mix-server is also required to prove that \(\log_u f_j = \log_g y_j\).

The method is naturally generalized to lists of El Gamal ciphertexts. Note that although the values \(e_j = \prod_{l \neq j \in T} -\left(u^{x_j} \right)^{-1} \) may be precomputed, it still requires an additional \(t\) number of exponentiations on the part of each mix-server, to decrypt a cryptotext.

A simple modification of the method above, is the following. The \(j\):th mix-server computes \(y_j' = y_j e_j'\) for \(j' \neq j\). Then it computes \(f_j = u^{x_j} e_j\) and proves that \(\log_u f_j = \log_g y_j\). This allows any mix-server to compute \(u^x = \prod_{j=1}^k f_j\), using no additional exponentiations (this observation is not explicit in Golle et al. [11]). Another, more straightforward, method is that we may assume that \(x = \sum_{j=1}^k x_j\), and that each \(x_j\) is shared by the \(j\):th mix-server using a VSS-scheme, and recreate \(x_j\) if the \(j\):th server turns out to be dishonest.

The method of proving correct decryption of individual ciphertexts is used twice in the second stage (decryption of inputs) of the protocol. Namely, first to decrypt the outer layer, and then to decrypt the inner layer.

### 6.1 Improved Analysis of the Original Protocol

Before we present our optimizations we observe that the complexity estimates presented in Fig. 1 of Golle et al. [11] in column “Proof and verification”, are overly pessimistic.

It seems that Golle et al. assume that the verification of each proof of knowledge requires 2 exponentiations. This is indeed true if a single proof is constructed and verified, but it is folklore knowledge\(^2\) that proofs of knowledge may be parallelized, which yields a verification cost of one exponentiation per proof.

Each mix-server must perform \((3 + 1)N\) exponentiations to jointly decrypt the cleartexts of the inner and outer layers. Each mix-server must also give \(4N\) proofs that she did so correctly, requiring \(4N\) exponentiations. Finally each mix-server must verify \(4(k - 1)N\) proofs, which requires \(4(k - 1)N\) exponentiations. This gives \(4(k + 1)N\) exponentiations in total. It seems fair to use this estimate in comparisons instead of the estimate \((5 + 10k)N\) from the original paper, since the required changes are very small.

### 7 Faster Joint Decryption

In this section we show that most of the proofs of knowledge used to ensure correctness of the decryption can be eliminated.

\(^2\)In fact Jakobsson suggested this to us! [18]
Using our modified protocol each mix-server only computes 1 proof of knowledge, and verifies \((k - 1)\) proofs during the decryption of the outer layer, giving a total cost of \(2(k - 1) + 1 = 2k - 1\) exponentiations for proving correctness of the outer decryption. It requires \(3N\) exponentiations to perform the actual outer decryption giving a total number \(3N + 2k - 1\) of exponentiations during the outer decryption.

We also show that if concurrent mix-sessions are not needed, i.e. if only a single mix-session is run using the same keys, we can remove the need for proofs in decryption of the inner layer completely. This is an obvious optimization if different keys are used for the inner and outer El Gamal systems, but can not be applied at all if identical keys are used. This idea was introduced in earlier work by Wikström [32].

Combined, our optimizations presented below give a mix-net in which each mix-server performs only \(4N + 2k - 1\) exponentiations during the decryption phase. The corresponding figure in the original protocol is \(4(k + 1)N\). The number of exponentiations performed by each mix-server in the re-encryption phase is unchanged from the original protocol (i.e. \(6N + 12k + 6\) of which \(6N\) may be precomputed).

For simplicity we assume in this section that the outer, and inner keys have been shared such that \(x_{\text{out}} = \sum_{j=1}^{k} x_{j,\text{out}}, y_{\text{out}} = g^{x_{\text{out}}} \), and \(y_{\text{in}} = \prod_{j=1}^{k} y_{j,\text{in}}\). But as explained above in Section 6, this can easily be generalized.

### 7.1 Faster Outer Joint Decryption

We observe that there is really no need to prove that the outer decryption is performed correctly \(\text{individually for each}\) El Gamal ciphertext. Instead each mix-server may simply prove that it performed the decryption correctly on the products, similarly as for a re-encryption.

Suppose that the output of the re-encryption stage in the protocol is given by the list: \(\{(\mu_i, 1, \nu_i, 2), (\omega_i, 1, \omega_i, 2)\}_{i=1}^{N}\). Then the modified outer decryption phase proceeds as follows:

1. Each mix-server computes the list: \(L_j = \{(f_j, \mu_i), (f_j, \nu_i), (f_j, \omega_i)\}_{i=1}^{N} = \{\mu_{x,j,\text{out}}, \nu_{x,j,\text{out}}, \omega_{x,j,\text{out}}\}_{i=1}^{N}\).

2. A quorum of mix-servers use Lagrange interpolation outlined above to compute the resulting list \(\{(u_i', v_i'), (u_i', w_i')\}_{i=1}^{N}\) of inner triples.

3. Each mix-server computes \(\omega_1 = \prod_{i=1}^{N} \omega_{i,1}\) and \(\omega_2 = \prod_{i=1}^{N} \omega_{i,2}\).

4. Each mix-server computes \(f_j = \omega_{j,\text{out}}^{x_{j,\text{out}}}\) and proves \(\log \omega_1 f_j = \log y_{j,\text{out}}\).

5. A quorum of the mix-servers to use Lagrange interpolation outlined above to compute \(w = D_{x_{\text{out}}} (\omega_1, \omega_2)\).

6. Each mix-server verifies that \(w = \prod_{i=1}^{N} w_i'\). If not, some mix-server cheated, and the original method, where correct decryption is proved for each individual El Gamal pair is used to identify the cheater (and then the back-up mix-net is invoked as in the original protocol).
7. A triple \((u'_i, v'_i, w'_i)\) is called valid if \(w'_i = h(u'_i, v'_i)\). To rule out that a mix-server cheated during the decryption phase, each mix-server is required to prove that \(\log_{\mu_{i,1}} f_{j,\mu,i} = \log_{\nu_{i,1}} f_{j,\nu,i} = \log_{\omega_{i,1}} f_{j,\omega,i} = \log_g y_{j,\text{out}}\), for all invalid triples.

8. Proceed as in the original protocol.

In fact what we have done above is to let the mix-servers jointly prove that the product of all inner triples, and all invalid triples are decrypted correctly. If that is the case, we know from the proofs of knowledge of the re-encryption stage that the products of the third components \(w_i = h(u_i, v_i)\) is unchanged from the input to the mix-net, and we may safely assume that invalid triples are either benign or the result of cheating during the re-encryption phase. This suffices to invoke the original results of Section 6 in Golle et al. [11].

The method described above optimizes the execution of a successful mix-session. In the original protocol each mix-server computes \(3N\) proofs of knowledge and verifies \(3(k-1)N\) proofs of knowledge during the joint outer decryption. This requires \(3kN\) exponentiations. Our modification reduces this to computing 1 proof of knowledge and verifying only \((k-1)\) proofs of knowledge, which requires \(2k-1\) exponentiations.

7.2 Faster Inner Joint Decryption

In this section we assume that the setting in which the mix-net is employed does not require several (concurrent) mix-sessions using the same set of keys. Under this assumption we observe that there is no need for the proofs of knowledge of the decryption of the inner layer at all.

The idea is very simple. If the mix-net is successful, each mix-server simply reveals its share \(x_{j,\text{in}}\) of the key to the inner El Gamal system, and verifies the keys revealed by the other mix-servers. This allows all mix-servers to compute the joint secret key of the inner El Gamal system, and then simply decrypt all elements, without using any proofs at all. This seems not to affect the security of the scheme, since if the mix-net is successful, the privacy does not depend at all on the secrecy of the secret key \(x_{\text{in}}\) of the inner El Gamal system.

A small number of exponentiations are possibly needed to verify the revealed keys, but that depends on the secret sharing scheme used.

If this optimization is used, new keys of the inner cryptosystem, must clearly be generated before any second mix-session takes place, and concurrent mix-sessions using the same keys are not allowed.

Our approach reduces the number of proofs of knowledge computed by each mix-server from \(N\) to zero, and the number of proofs verified by each mix-server from \((k-1)N\) to zero.

8 Conclusion

First we have presented two attacks on the elegant mix-net recently proposed by Golle et al. [11], claimed by the authors to be secure.

The first attack only requires that the adversary sends two messages to the mix-net to break the privacy of any given sender. Thus we break the privacy of the protocol completely.
The second attack may not be considered an attack at all, since it assumes that the protocol executes in a way that, although not clearly stated in the paper, is not the intended by the authors [12]. In addition to the first requirement on the execution of the protocol, the attack requires that the mix-net executes two mix-sessions using the same set of keys, and that the first mix-server is corrupted in the first of these mix-sessions. The attack shows the importance of explicitly defining details of the protocol.

We have also shown how to counter our attacks, but we do not claim that the modified protocol is secure.

Furthermore, we have presented modifications to the protocol that increase its efficiency. Assume that precomputation for the re-encryption phase is used and that the number of senders $N$ is large. Then our modifications reduce the number of exponentiations computed by each mix-server from $4(k+1)N$ to $4N$, where $k$ is the number of mix-servers. This means that our modified protocol outperforms all known mix-nets at least by a factor $k + 1$ for moderately large values of $N$, with complexity essentially independent of $k$.

This has important theoretical as well as practical consequences. It gives us a mix-net with linear complexity $O(N)$, also if the set of senders equals the set of mix-servers, i.e. each sender operates a mix-server. All previous mix-nets except Wikström [32] have complexity at least $O(N^2)$, but the mix-net by Wikström requires a very large number of senders.

Each re-encryption mix-net must at least decrypt the input, which for the ElGamal cryptosystem requires at least $N$ exponentiations. Thus it seems that our construction is close to optimal.

9 Future Work

We believe that the combination of the robustness test of Golle et al. [11] which is further developed in this work, and the notion of “double enveloping”, developed independently by both Golle et al. [11], and Wikström [32], may eventually give a provably secure, and efficient mix-net.

As explained above our modified mix-net is nearly optimal in terms of exponentiations executed by each mix-server, and it also seems secure. However, there is a relatively large number of attacks for mix-nets presented here, and in other papers [28, 28, 20, 5, 21], and there are still unclear details of the protocol presented here.

We conclude that future work on mix-nets should focus on explicitly defining the details of the protocol, and giving formal proofs, since applications such as electronic voting can otherwise not be considered seriously. To give formal proofs we must define a plausible security model in some security framework, e.g. that of Canetti [2], or that of Pfitzmann and Waidner [27], and define the notion of a mix-net formally. In future work we hope to give full proofs of security for the mix-net presented here.

10 Acknowledgements

Most of the results in this paper would have been far fetched without the many discussions with Gunnar Sjödin, Torsten Ekedahl, and Björn Grönvall on a
previous protocol by Wikström [32]. We are also very grateful to Johan Håstad for his support.

References


A Review of “Optimistic Mixing for Exit-Polls”

We give a short review of the protocol presented by Golle et al. [11]. To aid the reader most of the description given here is taken verbatim from [11], but it changes the notation slightly, and avoids some details that have no effect on our attacks. For details we refer the reader to the original paper [11].

The participants of the protocol are \( N \) senders, and a relatively small number \( k \) of mix-servers. The first set of participants perform some simple computations and write their input to the mix-net on a “bulletin board”. The second set of participants actually execute the mix-net protocol.

The mix-servers jointly generate parameters \((P, Q, g, x, y)\) for an El Gamal cryptosystem [7]. The El Gamal system is employed in a subgroup \( \langle g \rangle = G_Q \) of order \( Q \) of the multiplicative group modulo a prime \( P \), i.e. \( \mathbb{Z}_P^* \). Encryption of a message \( m \in G_Q \) using the public key \( y \) is given by \( E_y(m, r) = (g^r, y^r m) \), where \( r \) is chosen uniformly at random from \( \mathbb{Z}_Q \), and decryption of a cryptotext on the form \((u, v) = (g^r, y^r m)\) using the private key \( x \) is given by \( D_x(u, v) = u^{-x} v = m \).

The public parameters, i.e. \((P, Q, g, y)\), are made publicly known to all participants, but the secret key \( x \) is shared among the mix-servers using a threshold VSS scheme, where we let \( x_i \) denote the secret share of the \( i \)-th mix-server and \( y_i = x_i \).

One practical advantage of the mix-net is that it can be used to execute several mix-sessions using the same set of keys, i.e. the El Gamal keys are not changed between mix-sessions. To be able to do this the proofs of knowledge below are bound to a mix-session identifier \( id \) that is unique to the current mix-session.

A.1 Sending a Message to the Mix-Net

We first review what a typical honest sender Alice computes the following to send a message \( m \) to the mix-net:

\[
(u, v) = E_y(m), \quad w = h(u, v), \quad \text{and} \quad \alpha = (E_y(u), E_y(v), E_y(w)) = ((\mu_1, \mu_2), (\nu_1, \nu_2), (\omega_1, \omega_2)),
\]

where \( h \) is a hash function modeled by a random oracle. Then Alice computes a zero-knowledge proof of knowledge \( \pi_{id}(u, v, w) \) of \( u, v \) and \( w \), that depends on the current mix-session identifier \( id \). Finally Alice sends \((\alpha, \pi_{id}(u, v, w))\) to the “bulletin board”.

A.2 Execution of the Mix-Net

First the mix-servers remove any duplicate inputs to the mix-net, and discard input tuples that contain components not in the subgroup \( G_Q \). Then the mix-servers discard all input tuples where the proof of knowledge is not valid for the current mix-session. The mixing then proceeds in the following stages.

A.2.1 First Stage: Re-Randomization and Mixing.

This step proceeds as in all re-randomization mix-nets based on El Gamal. One by one, the mix-servers randomize all the inputs and their order. (Note that
the components of triples are not separated from each other during the re-
randomization.) In addition, each mix-net must give a proof that the product
of the plaintexts of all its inputs equals the product of the plaintexts of all its
outputs.

1. Each mix-server reads from the bulletin board the list of triples corre-
sponding to re-encryptions of $E_y(u_i), E_y(v_i), E_y(w_i)$ output by the previ-
ous mix-server: $\{ (g^{a_i}, a_i y^{c_i}), (g^{b_i}, b_i y^{c_i}), (g^{c_i}, c_i y^{c_i}) \}_{i=1}^N$. (Note that even
if some servers have cheated, the ciphertexts can still be formatted like
that, provided that every component belongs to $G_Q$.)

2. The mix-server re-randomizes the order of the triples according to a secret
and random permutation. Note that it is the order of triples that is re-
randomized, and that the three components $E_y(u_i), E_y(v_i)$ and $E_y(w_i)$
that make up each triple remain in order.

3. The mix-server then re-randomizes each component of each triple indepen-
dently, and outputs the results: $\{ (g'^{a_i}, a'_i y'^{c_i}), (g'^{b_i}, b'_i y'^{c_i}), (g'^{c_i}, c'_i y'^{c_i}) \}_{i=1}^N$.

4. The mix-server proves that $\prod a_i = \prod a'_i$, $\prod b_i = \prod b'_i$, and $\prod c_i = \prod c'_i$.

A.2.2 Second Stage: Decryption of the Inputs.

1. A quorum of mix-servers jointly decrypt each triple of ciphertexts to pro-
duce the values $u_i, v_i, w_i$. Since the method used to this is irrelevant
to our attack we defer the description of how this is done in the original
paper to Section 7, where we show how this step can be optimized.

2. All triples for which $w_i = h(u_i, v_i)$ are called valid.

3. Invalid triples are investigated according to the procedure described below.
If the investigation proves that all invalid triples are benign (only senders
cheated), we proceed to Step 4. Otherwise, the decryption is aborted and
we continue with the back-up mixing.

4. A quorum of mix-servers jointly decrypt the ciphertexts ($u_i, v_i$) for all
valid triples. This successfully concludes the mixing. The final output is
defined as the set of plaintexts corresponding to valid triples.

A.2.3 Special Step: Investigation of Invalid Triples.

The investigation proceeds as follows. The mix-servers must reveal the path of
each invalid triple through the various permutations. For each invalid triple,
starting from the last server, each server reveals which of its inputs corresponds
to this triple, and how it re-randomized this triple. The cost of checking the
path of an invalid triple is three exponentiations per mix-server (the same cost
as that incurred to run one input through the mix-net). One of two things may
happen:

- **Benign case (only senders cheated):** if the mix-servers successfully pro-
duce all such paths, the invalid triples are known to have been submitted
by users. The decryption is resumed after the invalid triples have been
discarded.
- **Serious case (one or more servers cheated):** if one or more servers fail to recreate the paths of invalid triples, these mix-servers are accused of cheating and replaced, and our mix-net terminates without producing an output. In this case, the inputs are handed over to the back-up mixing procedure described next.

**A.2.4 Back-Up Mixing.**

The *outer-layer* encryption of the inputs posted to the mix-net is decrypted by a quorum of mix-servers. The resulting set of *inner-layer* ciphertexts becomes the input to a standard re-encryption mix-net based on El Gamal (using, for example Neff’s scheme described in [22]). At the end of this second mixing, the ciphertexts are finally decrypted to plaintexts, which concludes the mixing.