Reusable Anonymous Return Channels

Philippe Golle
Stanford University
Stanford, CA 94305, USA
pgolle@cs.stanford.edu

Markus Jakobsson
RSA Laboratories
Bedford, MA 01730, USA
mjakobsson@rsasecurity.com

ABSTRACT

Mix networks are used to deliver messages anonymously to recipients, but do not straightforwardly allow the recipient of an anonymous message to reply to its sender. Yet the ability to reply one or more times, and to further reply to replies, is essential to a complete anonymous conversation. We propose a protocol that allows a sender of anonymous messages to establish a *reusable anonymous return channel.* This channel enables any recipient of one of these anonymous messages to send back one or more anonymous replies. Recipients who reply to different messages can not test whether two return channels are the same, and therefore can not learn whether they are replying to the same person. Yet the fact that multiple recipients may send multiple replies through the *same* return channel helps defend against the counting attacks that defeated earlier proposals for return channels. In these attacks, an adversary traces the origin of a message by sending a specific number of replies and observing who collects the same number of messages. Our scheme resists these attacks because the replies sent by an attacker are mixed with other replies submitted by other recipients through the same return channel. Moreover, our protocol straightforwardly allows for replies to replies, etc. Our protocol is based upon a re-encryption mix network, and requires four times the amount of computation and communication of a basic mixnet.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption

General Terms

Security

Keywords

Anonymity, Mix Networks, Privacy, Return Address

WPES'03, October 30, 2003, Washington, DC, USA.
Copyright 2003 ACM 1-58113-776-1/03/0010 ...$5.00.

1. INTRODUCTION

A private communication channel allows anonymous communication between parties. For example, it allows Alice to send an anonymous message to Bob. But an anonymous exchange is rarely limited to a single message. Often, Alice may want to allow Bob to reply to her anonymous message, and may then want to reply to Bob's reply, etc. In this paper, we propose the first practical protocol that allows two parties to have a complete conversation in an anonymous fashion.

The typical implementation of a private communication channel relies on a cryptographic primitive called mix network or mixnet, first proposed by Chaum [1]. A mixnet takes as input a list of ciphertexts and outputs the list of the corresponding plaintexts permuted in a random order. The mixnet essentially hides the relationship between its inputs and its outputs. Mixnets have been used for applications that require privacy, ranging from anonymous emails to anonymous voting, anonymous payments, etc.

A mix network lets Alice send an anonymous message to Bob as follows. Alice encrypts her message (including Bob's email address) under the public key of the mixnet. After the mixnet has collected a sufficiently large batch of ciphertexts submitted by different senders, it decrypts (or re-encrypts) all these ciphertexts and delivers them (in a random order) to their recipients. The identity of the sender of the message is hidden from the recipient of the message. Unfortunately, plain-vanilla mix networks do not permit anonymous replies. Some extensions to the basic mixnet design allow for one single anonymous reply without compromising the privacy of the exchange. The limitation to a single reply, however, is often too restrictive in practice. Other extensions allow for multiple replies, but are vulnerable to traffic analysis based on message counts: an adversary can trace the origin of a message by sending a specific number of replies and observing who collects the same number of messages.

The contribution of this paper is to design an extension of a synchronous mixnet that allows a sender of anonymous messages to establish a *reusable anonymous return channel.* This channel allows any recipient of a message originating with that sender to send back one or several anonymous replies. We now define more precisely the functional properties of anonymous return channels:

1. **Composable:** our protocol permits anonymous replies, as well as replies to replies, etc.

2. **Reusable:** a sender may use the same return channel for different messages sent to different recipients. The

recipients can not test whether their return channels are the same, yet they can each use the return channel to send one or several anonymous replies. As already noted, this helps defend against counting attacks.

3. **Transferable:** the ability to reply to a message is transferable in the sense that the recipient of a message can pass the message (and the ability to reply to it) on to someone else without leaking any private information other than the message itself. A consequence of this is that many different people may reply to the same message, if for example the original message was posted to a public bulletin board.

4. **Compatible:** our protocol processes replies in the same way as original messages during the mixing phase. Replies are only handled differently from original messages in a pre-processing step prior to mixing. As a result, messages cannot be distinguished from replies (or replies to replies) once the mixing has begun. This improves the privacy offered by the scheme.

1.1 Definition of Privacy

We turn now to the security properties of anonymous return channels, and start with a definition of our adversarial model. Recall that our system consists of a number of users who send messages to one another via a mixnet. The mixnet itself consists of a number of mix servers that process batches of messages sequentially.

We make the assumption common in the mixnet literature that the adversary has the ability to eavesdrop passively on any link in the network (between a user and a mix server, as well as between two mix servers). The adversary may not delete or corrupt messages in transit (in particular we do not consider denial-of-service attacks). The adversary may, however, statically control any number of users (who may submit messages of their own) and up to a minority of mix servers. Recall that giving the adversary static control means that the adversary must choose the subset of entities it wants to control before the protocol begins. This choice can not later be changed during the execution of the protocol. The adversary's computational resources are bounded by a polynomial in the security parameter.

The intuitive security requirement for our protocol is that it should protect the anonymity of all *honest* users against the adversary. We now formalize this requirement. We define privacy first for original messages, then for replies.

Original messages. Recall that messages are processed in batches. Let us consider a particular batch B. Let S (for sender) be the set of all users who submitted at least one message to B. We define the subset $S_0 \subseteq S$ of honest senders (i.e. those not controlled by the adversary). Let D be the distribution on S_0 that is proportional to the number of messages submitted by each sender in S_0 to the batch B.

Definition 1.1. **Privacy (original messages).** *We say that our protocol preserves the privacy of honest users if the adversary can not trace any original message in the output of the mixnet back to a sender in S_0 with accuracy greater than by drawing from S_0 at random according to distribution D.*

Observe that this definition of privacy for original messages is asymmetric with respect to senders and receivers. The identity of a honest sender is hidden among the set S_0 of all possible senders in that batch, regardless of whether the recipient of the message is controlled by the adversary or not. On the other hand, naturally, the anonymity of the receiver of an original message is only protected so long as the message was sent by a honest sender. In other words, the privacy of a receiver is conditional on the sender not being controlled by the adversary. This shouldn't surprise us: we are only rephrasing the fact that Alice by necessity must know Bob's identity since she initiated the communication with him.

Replies. Let us now consider what happens when "anonymous" messages are exchanged back and forth between Alice and Bob. Assume that Alice initiates an exchange with Bob, who then uses the anonymous return channel set up by Alice to reply to her. The definition of privacy for replies is the opposite of what it was for original messages. We can not prevent Alice (the recipient of the reply) from learning the identity of Bob (the sender), but we must ensure that Bob does not learn the identity of Alice. Let us again consider a particular batch B'. Let R (for receiver) be the set of all users who received at least one reply from B'. We define the subset $R_0 \subseteq R$ of honest receivers (i.e. those not controlled by the adversary). Let D' be the distribution on R_0 that is proportional to the number of replies received by each receiver in R_0 from the batch B'.

Definition 1.2. **Privacy (replies).** *We say that our protocol preserves the privacy of honest users if the adversary can not trace any reply in the input of the mixnet to a receiver with accuracy greater than by drawing from R_0 at random according to distribution D'.*

The identity of honest recipients is hidden among the set R_0 of all possible receivers, regardless of whether the sender of the reply is controlled by the adversary or not.

1.2 Traffic Analysis Attacks

We give more detail here about the threat presented by traffic analysis attacks (or counting attacks) and explain what makes our protocol resistant to these attacks. Assume that Alice has sent an anonymous message to Bob and has given Bob the ability to reply to her. In order to learn Alice's identity, Bob may send a specific number of replies in a given batch and observe who collects the corresponding number of messages in the output of the mixnet. Alternately, Bob may send replies at specific intervals, and observe who collects messages in the output of the mixnet in the same time periods. These are known as counting attacks.

Counting attacks are very hard to mount against our protocol for the following reason. After establishing an anonymous return channel with Bob, Alice may reuse this same return channel for her communications with Charlie, Dave, etc. Bob may well try to send a specific number of replies in a given batch, but since he can only observe the total number of messages collected by any player (and does not know which among these were sent by him) he won't be able to determine which player picked up his replies. This assumes naturally that Alice constantly receives messages or replies from multiple sources, and that Bob does not collude with every one of Alice's correspondents! In order to counter a counting attack that stretches over time (i.e., correlates input timings to timings of delivered messages) we allow either correspondent to alter the public key used to send the next reply – the other party will not know when this occurs, nor what key is replaced with what other key.

The threat of traffic analysis attacks can be further mitigated by techniques discussed in section 4, that allow for example a sender to set an upper bound on the maximum number of replies, or specify a deadline after which replies are no longer accepted. Finally, generic techniques borrowed from asynchronous mixnet designs, such as delaying messages or introducing dummy traffic, can help to reduce further the threat of traffic analysis attacks. We discuss these techniques in more detail below.

1.3 Related Work

Chaum [1] describes a technique that allows the recipient of an anonymous message to send a single anonymous reply with an "untraceable return address". This technique is vulnerable to traffic analysis attacks if multiple replies are allowed to be sent using the same return address. Indeed, the recipient of the replies risks being identified by the number of messages that he received through a particular return address. To allow for multiple replies, the sender must provide the recipient with multiple return addresses (one per reply), at a communication cost linear in the number of replies expected. Type I Cypherpunk Remailers [15], Babel [7], Onion routing [6] and Mixminion [3] all use variants of the same technique with the same essential limitation.

Type I Cypherpunk remailers and Babel however do allow for reuse of return addresses, but with severe limitations. In Type I remailers, the return address is not delivered directly to the recipient of a message but instead it is hidden behind a layer of indirection: a nymserver holds the return address on behalf of the recipient and is trusted not to reveal it to anyone. The obvious limitation of this approach is its reliance on trusted nymservers. Babel allows servers to rewrap replies with a few extra hops, called inter-mix detours. This technique ensures that replayed messages follow different paths, thus making them harder for an adversary to recognize. This and other asynchronous techniques can be used to strengthen any scheme, ours included.

The unique strength of our protocol, however, comes from the fact that our return channels are *reusable*. A sender can set up a single anonymous return channel and use it with an arbitrary number of correspondents. As already explained, the fact that numerous correspondents use the same return channel mitigates the risk of counting attacks. The privacy of all senders and recipients is guaranteed as long as each recipient receives messages from multiple sources, at least one of which is not controlled by the adversary.

What makes it possible for our return channels to be reused is the guarantee (based on the semantic security of ElGamal encryption) that two recipients can not determine whether their return channels are the same or not, and therefore do not know whether they are replying to the same person. In contrast, reply blocks based on Chaum's technique can not be reused with different players for if they were these players could simply compare their reply blocks and learn that they are communicating with the same person.

As already noted, generic techniques borrowed from asynchronous mixnet designs can help to reduce further the threat of traffic analysis attacks. In a synchronous mix network, all inputs are processed in batches, and elements belonging to one batch are processed together as they pass through the mix network. In an asynchronous mix network, on the other hand, messages may be separated from the rest of their batch in order to make timing based attacks more difficult. A drawback of such an approach is the increased difficulty of providing robustness, i.e., guarantees of correct mixing. Our work blurs the lines between synchronous and asynchronous mix networks to some extent. While our construction is decidedly synchronous in operation (and does offer robustness), it is clear that anonymity can benefit from some techniques normally associated with asynchronous mixing. In particular, introducing chaff elements in the input would help defend against traffic analysis based on the number of deliveries to various recipients. These chaff elements could correspond to replicated entries. One could also process all the inputs in a batch, but only deliver some portion of the outputs, entering the remainder into a "second loop" of the mix network. This corresponds to the notion of pool mixes, introduced by Cottrell [2].

Finally, in concurrent and independent work, Waters et al.[16] propose a protocol that allows the recipient of a message to reply multiple times to the sender by constructing "incomparable public keys". The adversary can not tell whether two such public keys belong to the same sender. A drawback of this approach is that it does not allow for direct deliveries of replies. Instead, replies must be broadcast to the set of all possible recipients, and recipients must sift individually for replies that are addressed to them. While this prevents any form of counting-based traffic analysis, it is not a practical technique, especially when the volumes are high as is desirable in any mix network.

Organization of the paper.
In the next section, we give some background on re-encryption mixnets and describe techniques for mixing ciphertext tuples and for switching encryption keys. In section 3, we describe a basic mixnet with reusable anonymous return channels. In section 4, we discuss how the sender of a message can set various kinds of limits on the use of the return channel. We conclude in section 5.

2. MIX NETWORKS PRELIMINARIES

We start this section with a quick overview of re-encryption mix networks. Readers unfamiliar with re-encryption mixnets may consult [13] for more detail.

For concreteness, we base our presentation of re-encryption mix nets on an ElGamal implementation. ElGamal is a probabilistic public-key cryptosystem, defined by the following parameters: a group G of prime order q, a generator g of G, a private key x and the corresponding public key $y = g^x$. Plaintexts are in G and ciphertexts in $G \times G$. ElGamal is semantically secure under the assumption that the Decisional Diffie Hellman (DDH) problem is hard in the group G.

Mix networks exploit the fact that the ElGamal cryptosystem allows for *re-encryption* of ciphertexts. Given an ElGamal ciphertext C and the public key y under which C was constructed, anyone can compute a new ciphertext C' such that C and C' decrypt to the same plaintext. Furthermore, without knowledge of the private key x, one cannot test if C' is a re-encryption of C if the DDH problem is hard in G.

Definition 2.1. *Re-encryption mix networks (ElGamal implementation)*

- **Key generation***: all mix servers jointly generate the parameters of an ElGamal cryptosystem. The parameters G, g, y are made public, while the private key x is shared among the mix servers [14, 5].*

- **Batch generation***: users are invited to submit to the mixnet ElGamal encrypted inputs $E(m)$ using the parameters generated above. Users are also required to submit a proof of knowledge for the corresponding plaintext m (see [8]). We do not concern ourselves with the policy that govern the creation of a batch, but we assume that at some point the batch is ready for processing. At this point, the batch is mixed and then decrypted (see below).*

- **Mixing phase***: Each server in turn mixes and re-encrypts the set of messages in the batch. More precisely, mix server M_i receives as input the set of ElGamal ciphertexts output by mix server M_{i-1}. Server M_i permutes and re-randomizes (i.e. re-encrypts) all these ciphertexts, and outputs a new set of ciphertexts, which is then passed to M_{i+1}. Server M_i must also provide a proof of correct mixing (see e.g. [4, 12, 11]).*

- **Decryption phase***: a quorum of mix servers jointly perform a threshold decryption of the final output, and provide a zero-knowledge proof of correctness for decryption.*

The mixnet with reusable anonymous return channels that we define in the next section differs slightly from the standard re-encryption mixnet given above. In addition to the fact that our new mixnet implements return channels of course, the two main difference are as follows:

Mixing ciphertext tuples

The inputs submitted by users to the mixnet will not consist of a single ElGamal ciphertext. Instead, we will require users to submit vectors of ElGamal ciphertexts (pairs, triplets, etc.) The number of ciphertexts in a vector will always be the same for all users, but it may be greater than one. Mix servers must re-encrypt individually every ciphertext component of the vector, then mix the vectors. Note that the vectors are mixed, but the order of the ciphertexts *within* any given vector is fixed.

One simple way to achieve this when the input elements are all ElGamal ciphertexts encrypted under the same public key is to generate a checksum for each vector of input ciphertexts. This checksum guarantees the integrity of the vectors: specifically the checksum ensures that elements of the same vector are not separated. The checksum is appended to the vector to form an augmented vector. We mix all the augmented vectors using a robust mix network and verify that the checksums in the resulting augmented output vectors are all correct. The checksum portion of the output vector can then be discarded.

To give a concrete example, the checksum may be computed as a product of the vector elements. Consider a vector $[(a_1, b_1), (a_2, b_2), (a_3, b_3)]$ of 3 ElGamal ciphertexts. The corresponding checksum is the ciphertext $(a_4, b_4) = (\prod_{i=1}^{3} a_i, \prod_{i=1}^{3} b_i)$. The mixing proceeds as follows. The first elements of every vector are all mixed together; the second elements of every vector are also mixed together, and so on. The mix servers reuse the same random permutation each time, but use random and independent re-encryption factors. The robustness of the mixing guarantees that vector elements have not been reordered. The checksum guarantees that vector elements have not been separated.

The correctness of the checksums may be verified either after every round of mixing, or just once at the end after every server has mixed the input vectors. To verify the checksums, we compute a new checksum for each output vector and compare this new checksum with the original checksum (which has accompanied the vector through the rounds of mixing). This comparison can be performed using a Plaintext Equivalence Test [10]. If all comparisons result in equality, the mixing is considered correct. If not, all servers involved in the mixing must reveal their re-encryption factors to prove they operated correctly.

A more efficient approach is to let each mix server prove plaintext equivalence by publishing an exponent α (for each tuple) such that if the *recomputed* output checksum ciphertext is re-encrypted with that exponent, one obtains the actual output checksum. This Plaintext Equivalence Test relies on the re-encryption factors of the server rather than the secret key of the mix network. It may be of particular use if the verification is to be performed after each round of mixing.

With either approach, the computational cost of mixing tuples is proportional to the number of elements in the vector.

Switching encryption keys

The outputs of the mixnet after the mixing phase are ElGamal ciphertexts encrypted under the public key of the mixnet. Recall that the corresponding private key is shared among all mix servers. These ciphertexts can not be delivered as such to recipients, because recipients do not know the private key of the mixnet and could therefore not decrypt the messages they receive. It is not acceptable either for the mixnet to decrypt the messages before delivering them: doing so would enable senders to recognize messages in the output and trace them to their recipients.

What we need is a technique that allows a quorum of mix servers to re-encrypt output ciphertexts under the public keys of the recipients to whom they are addressed. “Switching encryption keys” ensures that the ciphertext delivered to a recipient is encrypted under the public key of that recipient rather than under the public key of the mixnet. A correct, private, robust, and publicly verifiable method for switching encryption keys is described in [9]. We refer the interested reader to that paper for the details of the protocol.

3. REUSABLE ANONYMOUS RETURN CHANNELS

For simplicity, we base our initial presentation of reusable anonymous return channels on the following scenario. Alice sends an anonymous message M to Bob and wants to allow Bob to send her one or several anonymous replies $N_1, N_2, \ldots$ without revealing her identity. We show later that the same protocol also allows Bob to transfer to someone else the ability to reply to Alice.

All messages between Alice and Bob pass through a mix network to ensure privacy. Our protocol processes messages and replies in (almost) exactly the same way. For clarity however, we will describe first how Alice sends her original message M to Bob. Next we describe how Bob can reply anonymously to Alice.

Setting up the mixnet
The mix servers jointly generate the public and private parameters of an ElGamal cryptosystem. The public encryption parameters (the group G, the generator g and the public key y) are published, while the private key x is shared among the mix servers in a threshold manner [14, 5]. Let $E_{\mathcal{M}}$ and $D_{\mathcal{M}}$ denote the encryption and decryption functions for the ElGamal cryptosystem set up by the mixnet. The mix servers also establish a shared signing key. Let $S_{\mathcal{M}}$ and $V_{\mathcal{M}}$ denote the corresponding signing and verification algorithms. Note that there are robust algorithms for distributed decryption and signing, such that a minority of cheating servers can not produce an incorrect decryption or signature.

Submission of messages
The mixnet collects a batch of messages submitted by users over a certain time period. We describe here how Alice submits one message M to the mixnet. Naturally, Alice may submit several messages, and the mixnet also accepts messages (or replies, as described below) from other users.

We assume that Alice wants to send anonymously a message M to Bob. Let T_A be the identifying tag associated with Alice (her email or IP address for example), and let PK_A be Alice's public key. Note that Alice may use a different identifying tag or a different public key for different messages, but she need not do so. In fact, as discussed below, Alice's privacy is enhanced by the use of long lived tags and public keys.

Let PK_B denote Bob's public key, and T_B the identifying tag associated with Bob. Alice submits to the mix network the triplet

$$\Big(E_{\mathcal{M}}(T_A||PK_A);\;\; E_{\mathcal{M}}(M);\;\; E_{\mathcal{M}}(T_B||PK_B)\Big)$$

and proves knowledge of $(T_A||PK_A)$ and $(T_B||PK_B)$ to the mixnet[1]. If both proofs of knowledge are correct, the message is accepted.

Mixing the batch
When a batch is "ready" (e.g. when it contains enough message), the messages in the batch are mixed in turn by each mix server. Recall that each message in the batch consists of a triplet of ciphertexts. Each mix server in turn mixes and re-encrypts the triplets in the batch, in such a way that the respective ordering of elements within each triplet is preserved (see Section 2).

Delivery of messages
The mixnet outputs a list of triplets

$$\Big(E_{\mathcal{M}}(T_{S_i}||PK_{S_i});\;\; E_{\mathcal{M}}(M_i);\;\; E_{\mathcal{M}}(T_{R_i}||PK_{R_i})\Big),$$

where S_i is the sender of message M_i to recipient R_i for all $i = 1, \ldots, k$ (k is the number of messages in the batch). Each triplet in the batch is processed and delivered as follows. The mix servers jointly generate a signature Sig_i on the value $E_{\mathcal{M}}(T_{S_i}||PK_{S_i})$. The mix servers jointly decrypt the last element of the triplet to produce $(T_{R_i}||PK_{R_i})$. The mix servers then jointly convert $E_{\mathcal{M}}(M_i)$ into $E_{PK_{R_i}}(M_i)$ (see Section 2 for more detail on "switching" encryption keys). The message $E_{PK_{R_i}}(M_i)$ together with $E_{\mathcal{M}}(T_{S_i}||PK_{S_i})$ and the signature Sig_i are delivered to the recipient (as identified by the tag T_{R_i}). The recipient can decrypt that message to recover the plaintext M_i. The recipient keeps the value $E_{\mathcal{M}}(T_{S_i}||PK_{S_i})$ and the signature Sig_i. These will be necessary to reply to M_i (see below)

Submitting a reply
We assume that Bob has received the message M, along with the value $E_{\mathcal{M}}(T_A||PK_A)$ and the signature Sig of the mixnet on that value. The submission of a reply message to the mixnet is nearly identical to the submission of an original message. Let N denote the content of Bob's reply. Bob must in turn supply a public key (encrypted under $E_{\mathcal{M}}$) which Alice may use should she want to reply to Bob's reply. This public key may be (but need not be) the public key PK_B of Bob that Alice used to send her original message. Bob may also use a different key. Either way, let PK_B denote the public key Bob uses for his reply N. Bob submits to the mixnet the triplet

$$\Big(E_{\mathcal{M}}(T_B||PK_B);\;\; E_{\mathcal{M}}(N);\;\; E_{\mathcal{M}}(T_A||PK_A);\;\; Sig\Big)$$

and proves knowledge of $(T_B||PK_B)$ only. The message is accepted if that proof of knowledge is correct *and* the signature Sig on the tag $E_{\mathcal{M}}(T_A||PK_A)$ is correct. From that point on, replies are mixed and delivered in exactly the same way as original messages.

A note on the use of signatures to guarantee the authenticity of tags:
The mixnet signature Sig on the tags $E_{\mathcal{M}}(T_A||PK_A)$ ensures that the users do not tamper with tags. Since tags are eventually decrypted by the mixnet (in the delivery phase), this precaution is crucially important to prevent users from using the mixnet as a decryption oracle on any ciphertext of their choice encrypted under the mixnet's public key.

Observe that for the purpose of guaranteeing the integrity of tags, a message authentication code (MAC) would do just as well as a signature. MACs may be computed more efficiently than signatures, but there is a caveat. While a signature can be generated and verified jointly by any quorum of mix servers (each of whom holds a share of the private key), the same is not possible with a MAC. If the authenticity of the tag $E_{\mathcal{M}}(T_A||PK_A)$ is to be guaranteed by a MAC, each server will have to apply its MAC to the tag. Later, each server can individually verify its own MAC on the input tag submitted by a user. A server cannot verify any other server's MAC. The drawback of this solution then is that it requires as many MACs as there are mix servers (versus a

[1] Proofs of knowledge are important for two reasons. First, they prevent users from resubmitting ciphertexts. Second, they prevent users from using the mixnet as a decryption oracle for ciphertexts encrypted under the public key of the mixnet. In these proofs of knowledge, the verifier (i.e. the mixnet) should issue challenges to the user that depend on the value $E_{\mathcal{M}}(M)$ to avoid splicing attacks. See [8] for more details.

single signature) and also requires a large overlap between the servers that generate and verify the MACs (which implies that the reply must be processed by the same servers that processed the original message).

Properties:

- **Composable.** Our protocol allows straightforwardly for replies to replies, as well as replies to replies to replies, etc.
- **Reusable.** Bob can reuse the value $E_{\mathcal{M}}(T_A||PK_A)$ and the corresponding signature Sig to send as many replies as desired. All replies sent by Bob are anonymous. Alice may also reuse the same reply tag in communication with other parties.
- **Transferable.** Bob may pass on the tag $E_{\mathcal{M}}(T_A||PK_A)$ and the accompanying signature Sig to enable any other user to reply to Alice. For that matter, Bob need not be an individual user but could also be a public bulletin board (in which case encryption using the public key of the receiver PK_B may be the identity function, effectively making the output message publicly readable after the switching of encryption keys).
- **Compatible.** The mix network processes replies and original messages in almost exactly the same way. The only difference is that at submission time, an original message requires two proofs of knowledge whereas a reply requires one proof of knowledge and one signature verification. This admittedly allows an adversary to distinguish replies from original messages, but only at the time of input. During the mixing phase, one cannot tell an original message from a reply. The ability to mix original messages and replies together in the same batch not only enhances privacy, but also simplifies the design of the anonymous communication system.
- **Efficiency.** Our mixnet has four times the computational and communication cost of a standard plain-vanilla mixnet.
- **Threat.** As discussed in the introduction, our protocol for return channels may be vulnerable to traffic analysis attacks. These attacks work best when the volume of replies is very low. If we assume a high volume of replies, the threat presented by these attacks is minimal. These attacks can be further mitigated by the techniques discussed in the next section.

4. BOUNDING REPLIES

It may sometimes be desirable to place bounds on the use of return channels, both as a privacy control technique (as discussed above) and out of practical considerations. For example, a seller who advertises an item may be interested in no more than a particular number of replies, or may not want to receive replies past a certain deadline. In this section, we propose return channels that can be used only before a certain date, or only a specific number of times, and are then rejected if input to the mix network.

Throughout, we use the notation F_M for the filtering information submitted by the sender of a message M. The filtering information F_M may specify the maximum total number of replies to the message, or the deadline past which replies are disallowed, or the maximum number of replies allowed per time period, or any combination of these policies. Observe however that while all these policies are possible, some are entirely stateless (e.g. a deadline past which replies are disallowed), while others require the mixnet to keep track of a potentially very large amount of information (e.g. bounding the number of replies).

We introduce two techniques for bounding replies. The first provides the convenience described above but does not help defend against traffic analysis attacks. The second technique helps defend against traffic analysis attack. We call Alice the sender of the message and Bob its recipient, but note that our techniques for bounding replies work equally well for original messages and replies.

4.1 Output Filtering

The sender of a message M (Alice) may include the filtering policy F_M associated with that message in the first element $E_{\mathcal{M}}(T_A||PK_A)$ of the triplet. Thus Alice submits to the mixnet a triplet formatted as:

$$\Big(E_{\mathcal{M}}(T_A||PK_A||F_M);\;\; E_{\mathcal{M}}(M);\;\; E_{\mathcal{M}}(T_B||PK_B)\Big)$$

Bob's reply to that message will be of the form:

$$\Big(E_{\mathcal{M}}(T_B||PK_B||F_N);\;\; E_{\mathcal{M}}(N);\;\; E_{\mathcal{M}}(T_A||PK_A||F_M);\;\; Sig\Big)$$

where the value F_N is optional (Bob need not specify a filtering policy for his reply to Alice).

The mix network mixes and re-encrypts all triplets as in the original protocol. As in the original protocol, the mixnet decrypts the last element of the triplet to produce $(T_A||PK_A||F_M)$. Here, the mixnet verifies that Bob's reply to M is allowed by the filtering policy F_M. If not, the mixnet drops the triplet. If the reply is allowed, the mixnet proceeds as before. It converts $E_{\mathcal{M}}(N)$ into $E_{PK_A}(N)$ and sends to Alice $E_{PK_A}(N)$ along with $E_{\mathcal{M}}(T_B||PK_B||F_N)$ and a signature Sig on that value.

Properties:
Note that output filtering limits replies forwarded to receivers but it does not help defend against traffic analysis attacks. Indeed, the filtering policy is decrypted together with the recipient of the message. The filtering policy is thus ineffective against an adversary who controls at least one mix server: a message that is disallowed by the filtering policy can not be discarded before the adversary has learned its intended recipient.

4.2 Input Filtering

In this approach, the mixnet verifies that a message is allowed by the appropriate filtering policy at the time the message is submitted. The message is only mixed and delivered if the filtering policy allows it. Other messages are discarded even before being mixed. We now describe the technique in more detail.

The filtering policy F_M is encrypted under the public key of the mixnet and submitted together with the rest of the message. Thus messages are formatted as quadruplets instead of triplets:

$$\Big(E_{\mathcal{M}}(T_A||PK_A);\;\; E_{\mathcal{M}}(M);\;\; E_{\mathcal{M}}(T_B||PK_B);\;\; E_{\mathcal{M}}(F_M)\Big)$$

Assume for now that there is no filtering policy defined for the recipient indicated by the tag $E_{\mathcal{M}}(T_B||PK_B)$ and that the message is accepted. The mix servers mix and re-encrypt these quadruplets as in the original protocol. The only difference is that the mix network outputs a signature on the value

$$E_{\mathcal{M}}(T_A||PK_A);\ \ E_{\mathcal{M}}(F_M)$$

The mixnet sends to Bob the message $E_{PK_B}(M)$, as well as the values $E_{\mathcal{M}}(T_A||PK_A)$ and $E_{\mathcal{M}}(F_M)$ and the signature Sig. The mixnet may optionally decrypt $E_{\mathcal{M}}(F_M)$ and send the plaintext F_M to Bob, so that the recipient is aware of the filtering policy associated with M.

Bob may reply to this message with the sextuplet

$$\Big(E_{\mathcal{M}}(T_B||PK_B);\ \ E_{\mathcal{M}}(N);\ \ E_{\mathcal{M}}(T_A||PK_A);$$
$$E_{\mathcal{M}}(F_N);\ \ E_{\mathcal{M}}(F_M);\ \ Sig\Big)$$

When this sextuplet is input, the mixnet verifies the signature Sig on the value

$$E_{\mathcal{M}}(T_A||PK_A);\ \ E_{\mathcal{M}}(F_M)$$

Bob's reply is discarded if the signature is incorrect. If the signature is correct, the mixnet decrypts $E_{\mathcal{M}}(F_M)$ and checks whether Bob's reply is allowed by the policy F_M. If not, Bob's reply is discarded. If it is allowed, the mixnet strips the sextuplet input by Bob of $E_{\mathcal{M}}(F_M)$ and of Sig. The resulting quadruplet is processed as described above.

Properties:
Input filtering helps defend against traffic analysis attacks. Messages that are not allowed by the filtering policy are discarded and the adversary can not learn the recipient for whom they were intended. On the downside, input filtering comes at a higher computational cost since it requires mix servers to process quadruplets rather than triplets.

5. CONCLUSION

We propose the first protocol that allows for reusable anonymous return channels. Our protocol makes possible flexible complete anonymous conversations. The main advantages of our protocol are as follows. The same return channel can be reused to allow any number of recipients to reply to a sender. These recipients can not test whether or not they are replying to the same person. The protocol allows for replies, as well as replies to replies, etc. The protocol processes replies and original messages in almost the same way and the ability to reply to a message is transferable. Our protocol has four times the computational cost of a basic re-encryption mixnet. In practice, this translates into acceptable performance.

The ability to reply multiple times to an anonymous message introduces potential new threats to privacy. The receiver of an anonymous message may try to learn the identity of the sender by sending numerous replies and observing who collects these replies. We describe techniques to mitigate this risk by allowing senders to specify limits on how many replies they are willing to accept.

6. ACKNOWLEDGMENTS

The authors would like to thank Roger Dingledine for his help in identifying related work, as well as the anonymous referees for their helpful comments.

7. REFERENCES

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

[2] L. Cottrell. Mixmaster & remailer attacks. `http://www.obscura.com/~loki/remailer/remailer-essay.html`, 1995.

[3] G. Danezis, R. Dingledine and N. Mathewson. Mixminion: design of type III anonymous remailer protocol. In *Proc. of the 2003 IEEE Symposium on Security and Privacy*, pp. 2-15.

[4] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Proc. of Crypto '01*, pp. 368-387. Springer-Verlag, 2001. LNCS 2139.

[5] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Proc. of Eurocrypt '99*, pp. 295-310. Springer-Verlag, 1999. LNCS 1592.

[6] D. Goldschlag, M. Reed and P. Syverson. Onion routing for anonymous and private internet connections. In *Communications of the ACM*, 42(2):39-41, 1999.

[7] C. Gulcu and G. Tsudik. Mixing E-mail with Babel. In *Proc. of Network and Distributed Security Symposium - NDSS '96*. IEEE, 1996.

[8] M. Jakobsson. A practical mix. In *Proc. of Eurocrypt '98*, pp. 448-461. Springer-Verlag, 1998. LNCS 1403.

[9] M. Jakobsson. On quorum controlled asymmetric proxy re-encryption. In *Proc. of Public Key Cryptography '99.*

[10] M. Jakobsson and A. Juels. Mix and match: secure function evaluation via ciphertexts. In *Proc. of Asiacrypt '00*, pp. 162-177. Springer-Verlag, 2000. LNCS 1967.

[11] M. Jakobsson, A. Juels and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proc. of USENIX '02*, pp. 339-353, 2002.

[12] A. Neff. A verifiable secret shuffle and its application to E-Voting. In *Proc. of ACM CCS'01*, pp. 116-125. ACM Press, 2001.

[13] W. Ogata, K. Kurosawa, K. Sako and K. Takatani. Fault tolerant anonymous channel. In *Proc. of ICICS '97*, pp. 440-444, 1997. LNCS 1334.

[14] T. Pedersen. A Threshold cryptosystem without a trusted party. In *Proc. of Eurocrypt'91*, pp. 522-526, 1991.

[15] S. Parekh. Prospects for remailers. *First Monday,* 1(2), August 1996. On the web at `http://www.firstmonday.dk/issues/issue2/remailers/`

[16] B. Waters, E. Felten and A. Sahai. Receiver anonymity via incomparable public keys. To be presented at the *2003 ACM Conference on Computer and Communications Security.*