Minx: A Simple and Efficient Anonymous Packet Format

George Danezis
University of Cambridge, Computer Laboratory,
William Gates Building, 15 JJ Thomson Avenue,
Cambridge CB3 0FD, United Kingdom.
George.Danezis@cl.cam.ac.uk

Ben Laurie
ALD Ltd,
The Stores, 2 Bath Road,
London W4 1LT, United Kingdom.
ben@algroup.co.uk

ABSTRACT

Minx is a cryptographic message format for encoding anonymous messages, relayed through a network of Chaumian mixes. It provides security against a passive adversary by completely hiding correspondences between input and output messages. Possibly corrupt mixes on the message path gain no information about the route length or the position of the mix on the route. Most importantly Minx resists active attackers that are prepared to modify messages in order to embed tags which they will try to detect elsewhere in the network. The proposed scheme imposes a low communication and computational overhead, and only combines well understood cryptographic primitives.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*

General Terms

Security

Keywords

anonymity, mix networks, tagging attacks

1. INTRODUCTION

In [10] Danezis *et al.* propose *Mixminion*, a mix-packet design that provides bitwise unlinkability and is resistant to all known active tagging attacks. Furthermore it supports anonymous replies, using a mechanism that makes them indistinguishable from other messages. The design is being standardised and implemented to replace the ageing Type I 'Cypherpunk' and Type II 'Mixmaster' remailer infrastructure [17].

Mixminion was designed to be well understood, secure and robust. As a result the design is conservative, including redundant information and mechanisms. Mixminion also aims to be flexible and, as a result, it is quite complex.

We propose *Minx*, a new mix-packet format design that is simpler, yet provides all the security properties that mixminion provides. Our design uses two cryptographic primitives, raw RSA [24] and AES [1], and IGE [26] a special mode of operation for block ciphers. Minx does not require Mixminion's 'swap step', which is still not very well understood. Instead it neutralises tagging attacks by using a block cipher mode that propagates errors: if the message is tagged all useful information is destroyed.

A heuristic security argument is provided, but the minimalist nature of the packer format is intended to foster research on more formal ways of proving the security of such anonymity mechanisms.

WPES'04, October 28, 2004, Washington, DC, USA.

2. REQUIREMENTS

The security requirements of Minx are the same as those for mixminion. This allows a straightforward comparison between the more conservative mixminion design and the new Minx proposal.

Minx is a mix packet format, to encode messages that will be relayed through a mix network as originally proposed by David Chaum [7]. A mix network is a set of nodes that relay messages without revealing the correspondence between their inputs and outputs, making it difficult to link their senders and receivers.

The main aim of a mix packet encoding is for the bit pattern of the messages entering the mix to be unlinkable to the bit patterns of messages leaving the mix. Additionally routing information must be privately delivered to all intermediate mixes to allow them to route the message through the network. We also aim to support anonymous return addresses that allow receivers of messages to contact the anonymous sender without knowing their physical address, by using an anonymous reply block.

We assume that an adversary observes all links of the network, and so can get access to the bit patterns and timings of all messages. Furthermore the attacker controls a subset of mixes on the path of each message. To minimise the information hostile nodes gain, we require our mix packet format to not leak the position of the mix in the path, and to not reveal whether the message is a reply or not.

Finally an attacker can be active, manipulate messages and re-inject them into the network. Such tagging attacks [22] are designed to leak information to the attacker by producing an observable result that links the final destination of the message to the message's sender. Minx eliminates such

attacks, at the cost of destroying the relayed message.

We can summarise the Minx design requirements as follows:

1. Anonymity despite a global passive adversary that also controls a subset of nodes on the message path, and can perform active attacks against the honest mix servers' networks.
2. A facility for secure anonymous replies, indistinguishable from other messages.
3. The position of a mix on the path and the path length should be hidden from intermediate mixes.
4. Tagging attacks must be totally eliminated.

In addition to the above security requirements we shall also require that only parties benefiting from anonymity properties are required to use special software. This makes deployment easier and provides appropriate incentives to users.

Like Mixminion and other deployed remailer networks we assume the existence of a directory service that provides a list of trusted mix nodes. We assume that the list, and the index numbers associated with each server and their public keys, are long lived. This affects the lifespan of reply blocks, but frequently changing lists and keys should not have a dramatic effect on the reliability of sender anonymous messages.

Note that the effects of directory updates can be mitigated by preserving the position of (most of the) existing entries and the length of the directory as a whole (by including redundant entries). While assuming that a directory service infrastructure is available, the details of how it might be implemented and secured are not discussed here (see [15] for details).

3. CRYPTOGRAPHIC PRIMITIVES

Each mix in the Minx system has an RSA [24] key pair. We will assume that the RSA modulus is 1024 bits long, and therefore all messages up to 1023 bits can be correctly encoded. We use 'raw' RSA (modular exponentiation), without any padding scheme, since we will rely on mixes not being able to recognise valid plaintext (opposite property from the plaintext awareness [12] provided by most padding schemes).

Important properties of Minx are provided by using the AES block cipher in a special mode of operation named Infinite Garble Extension (IGE) [26]. IGE is a variant of Cipher Block Chaining, that XORs the previous ciphertext and plaintext to each encrypted block. We denote $\mathcal{E}_k(\cdot)$ the AES encryption operation with key k, and $\mathcal{D}_k(\cdot)$ the respective decryption operation. If P_i is the i^{th} plaintext block and C_i the corresponding ciphertext block, IGE encoding can be described as:

$$C_i = \mathcal{E}_k(P_i \oplus C_{i-1}) \oplus P_{i-1} \tag{1}$$

The corresponding decoding relation is:

$$P_i = \mathcal{D}_k(C_i \oplus P_{i-1}) \oplus C_{i-1} \tag{2}$$

It is clear that if a block of ciphertext or plaintext is modified all subsequent blocks will decode into random noise. We shall be denoting IGE encoding of a message M by a key k, as $IGE_k(M)$ and decoding as $IGE_k^{-1}(M)$. Note that we assume that C_{-1} and P_{-1}, the initialisation vectors, are set to a well know value, such as a vector of zero bits. This does not represent a threat since in the context of Minx, since IGE and biIGE keys are only used once.

In some cases perfect forward propagation of errors is not sufficient, and backwards propagation of errors is additionally required. The IGE mode, presented above, can be applied twice on a plaintext M to achieve this effect. First it is applied to the plaintext, then the resulting ciphertext is reversed and a second encryption operation using the IGE mode is performed. This mode of operation is called bidirectional Infinite Garble Extension, or biIGE:

$$biIGE_k(M) = IGE_k(reverse(IGE_k(M))) \tag{3}$$

The property of interest of biIGE mode is its all-or-nothing nature: if the ciphertext is modified none of the plaintext can be recovered. Unlike the IGE mode that propagates errors only forward, biIGE propagates them to the whole plaintext or ciphertext. A similar effect could be achieved by using BEAR [2], a variable block length block cipher, as Mixminion does. Other 'all-or-nothing' transforms [**?**] can be used instead of BiIGE.

Although the properties Minx relies on are provided by IGE and biIGE, they are not specific to these schemes. Therefore we will be describing the Minx algorithms in terms of the abstract modes of operation EP (Error Propagation) and biEP (bidirectional Error Propagation). EP hides the correspondence between plaintexts and ciphertexts, and propagates any modifications of the plaintext and ciphertext forward, as IGE does. BiEP also propagates any modification backwards, as biIGE does.

The biEP mode can, of course, be generally constructed from EP just as biIGE is constructed from IGE:

$$biEP_k(M) = EP_k(reverse(EP_k(M))) \tag{4}$$

though this does assume that EP has the property that ciphertext blocks are ordered naturally.

An alternative construction for EP would be to use the existing PCBC mode, or use an HMAC [4] of the previous plaintext as the the key for the next block to be encrypted. This would be substantially less efficient than IGE (the key schedule would need to be recomputed for every block), but demonstrates the existence of other possible constructions.

4. THE BASIC PACKET FORMAT

An anonymous sender Alice wishes to encode and send a message M. This message contains all necessary information to reach the final recipient, such as an email address, the body of the message to be delivered, as well as information about the delivery mechanisms (M could be an RFC 821 [23] formatted email message including the headers and the body, with all identifying information removed).

Alice downloads the *current directory* of mix servers from a directory server she trusts. The directory is a simple list of currently running mix servers along with their public keys and addressing information. Each mix server entry is denoted by a small integer (8 bits would suffice) and all entries are filled, either by distinct servers or by replicating identical server entries for multiple entries, or you take the index number modulo the directory length at each step, then you can use a 4-byte index number in the message format.

Alice then chooses a number of mix servers out of this directory to relay her message M of fixed size l (she can pad M

up to length l). We will denote these servers by their respective directory indexes as $i_1, i_2, \ldots, i_n$ and their respective public keys $PubK_1, PubK_2, \ldots, PubK_n$. Alice then chooses n random session keys $k_1, k_2, \ldots, k_n$[1] and encodes her message as follows. (We denote the concatenation of bit-strings with "|", and the byte range from byte x to byte y of string s as "$s[x–y]$".)

$$\text{Encode}(M, i_1 \ldots i_n, PubK_1 \ldots PubK_n, k_1 \ldots k_n, l, tag = \text{Final}) : \quad (5)$$

$$P_n = k_n | tag | biEP_{k_n}(M) \quad (6)$$

$$C_n = RSA_{PubK_n}(P_n[0–127]) | P_n[128–] \quad (7)$$

$$\text{For } x \text{ from } n-1 \text{ to } 1 : \quad (8)$$

$$P_i = k_i | i_{i+1} | EP_{k_i}(C_{i+1}) \quad (9)$$

$$C_i = RSA_{PubK_i}(P_i[0–127]) | P_i[128–] \quad (10)$$

$$\text{Pad } C_1 \text{ up to a set size: } C_1 = C_1 | J_l \quad (11)$$

The padded ciphertext C_1 can be sent to the mix entry i_1, that will decode it and relay it to the other mixes which, in turn relay it until it reaches its ultimate destination. The decoding operation, that each mix j with private key PrK_j performs, is very simple:

$$\text{Decode}(PrK_j, M') : \quad (12)$$

$$k_j | i_j | encM = RSA_{PrK_j}(M'[0–127]) | M'[128–] \quad (13)$$

$$\text{Check and store } H_{id}(k_j). \quad (14)$$

$$\text{if } i_j \text{ is not "Final":} \quad (15)$$

$$M = EP^{-1}_{k_j}(encM) \quad (16)$$

$$\text{Send padded message } M|J \text{ to } i_j \quad (17)$$

$$\text{else:} \quad (18)$$

$$\text{Process message } biEP^{-1}_{k_j}(M[0–l-1]). \quad (19)$$

All packets are padded by the senders or the mixes by appending a random bit-string J_l of the appropriate size. Mixminion pads packets up to 32 kb, and Minx could safely do the same.

The "Final" tag indicates to a mix server that it is the last in the chain and the final message should be decoded and processed. As all the other directory indexes it is an integer with the property that $Final \mod 2^3 = 0$. Obviously all such indexes are reserved to represent the "Final" tag and cannot be used for mix servers. Note that if Alice picks the route by choosing random 8 bit integers the path length would follow a geometric distribution with parameter $\frac{1}{8}$, and the average path length would be 8.

5. REPLY BLOCKS

So far we have described how a user can encode a sender anonymous message, and how mix servers decode and forward messages. One of the aims of Minx is to provide the facilities required to reply to an anonymous senders. This is achieved by the use of *anonymous reply blocks*, which route the message from the sender to the anonymous receiver. An anonymous sender can generate a number of anonymous reply blocks and include them in a message to allow correspondents to reply.

[1]Each key k_j must be an integer such that the overall message can be decrypted by the public key $PubK_j$. i.e. $PubK_j$ divided by $2^{(1024-80)}$ is greater than k_j.

An anonymous reply block rb_{A} is encoded as a normal message with a very small payload containing information necessary to decode the rest of the message. This can be thought as a communication channel from the creator of the reply block, namely Alice, back to herself that allows her to be stateless.

As for encoding a message, to create a reply block Alice chooses a set of mixes with indexes $i_1, \ldots, i_n$ and respective public keys $PubK_1, \ldots, PubK_n$. A number of session keys are then generated from a master key MK such that $k_j = H(MK, j)$. The use of a master key allows for decoding reply messages without storing any state per message or reply block. Note that the length the message is going to be padded to is l', and much smaller than l, the total message length, and the the special label Reply is included as the routing information provided to the last node in the path.

$$rb_{\text{A}} = \text{Encode}(m_A, i_1 \ldots i_n, PubK_1 \ldots PubK_n, \quad (20)$$

$$k_1 \ldots k_n, l', tag = \text{Reply}) \quad (21)$$

where m_A is,

$$m_A = \text{Alice}|IV| \quad (22)$$

$$biEP_{H(K_{\text{Alice}}|IV)}(\text{Alice}|\text{Nym}|n|i_1|\ldots|i_n|MK) \quad (23)$$

The IV is a large nonce and K_{Alice} is Alice's secret key. The 'Nym' is the pseudonym associated with this reply block, and has to be present to foil some higher layer attacks. Two differences between generating sender anonymous messages and reply blocks should be noted: reply blocks are only padded to a much smaller length l', and the label seen by the last node on the path should be a different constant denoted "Reply". This allows the final node to decode, using biEP, only the necessary part of the message.

The above construction generates a small anonymous reply block, that Alice's recipients can use to reply to Alice. A recipient that wants to reply to Alice with reply block i, rb_{A} and message $\bar{M}$ simply constructs a message (The $NULL$ key is a well known constant key, such as the all zero key):

$$rb_{\text{A}}|biEP_{NULL}(\bar{M})|J \quad (24)$$

The message is then sent to the first mix i, that will decode and forward it like all other messages until the last mix. In case the responder does not have special software to perform this encoding, a proxy could be used. The last mix can recover that the message should be routed to Alice, and sends it to her, by email or other means.

6. DECODING ANONYMOUS REPLIES

Alice can decode, using her secret K_{Alice}, the information contained in the reply message M'', recover the intermediary hops $i_1, \ldots, i_n$ and the master key MK that generates the session keys $k_1, \ldots, k_n$, and reconstruct the reply block, and all other associated information:

$$\text{Alice}|IV|M_2 = M'' \quad (25)$$

$$\text{Alice}|\text{Num}|n|i_1|\ldots|i_n|MK = biEP_{H(K_{\text{Alice}}|IV)}(M2[l'']) \quad (26)$$

$$k_i = H(MK|i) \quad (27)$$

$$PubK_j = \text{lookput directory entry for } i_j \quad (28)$$

This allows Alice to reconstruct the reply block from information contained in the message, without any additional

state kept per reply block. Note that the exact reply block is necessary to decode the message $\bar{M}$ sent by Bob since, in the general case, the EP mode of operation relies on all previous plaintext blocks being known to recover the message.

$$m_A = \text{Alice}|IV| \quad (29)$$

$$biEP_{H(K_{\text{Alice}}|IV)}(\text{Alice}|\text{Nym}|n|i_1|\dots|i_n|MK) \quad (30)$$

$$rb_{\text{A}} = \text{Encode}(m_A, i_1 \dots i_n, PubK_1 \dots PubK_n, \quad (31)$$

$$k_1 \dots k_n, l', tag = \text{Reply}) \quad (32)$$

In the case of IGE additional information is necessary to decode the message, aside from the session keys. Since each IGE block, aside from the key, is dependant on both the previous plaintext and ciphertext blocks, we need to compute these. They can then be used a initialisation vectors (IV_{P_i} and IV_{C_i}), to decode the rest of the message:

$$\text{GetIVs}(m_{\text{A}}, i_1 \dots i_n, PubK_1 \dots PubK_n, k_1 \dots k_n, l', tag = \text{Reply}) : \quad (33)$$

$$P_n = k_n|tag|biEP_{k_n}(M) \quad (34)$$

$$C_n = RSA_{PubK_n}(P_n[0\text{–}127])|P_n[128\text{–}] \quad (35)$$

$$IV_{P_n} = M[-blocklength :] \quad (36)$$

$$IV_{C_n} = C_n[-blocklength :] \quad (37)$$

$$\text{For } x \text{ from } n-1 \text{ to } 1 : \quad (38)$$

$$P_i = k_i|i_{i+1}|EP_{k_i}(C_{i+1}) \quad (39)$$

$$IV_{P_i} = C_{i+1}[-blocklength :] \quad (40)$$

$$IV_{C_i} = P_i[-blocklength :] \quad (41)$$

$$C_i = RSA_{PubK_i}(P_i[0\text{–}127])|P_i[128\text{–}] \quad (42)$$

$$\text{return } IV_{P_n}, IV_{C_n}, \dots, IV_{P_1}, IV_{C_1} \quad (43)$$

Given the initialisation vectors $(IV_{P_n}, IV_{C_n}, \dots, IV_{P_1}, IV_{C_1})$ the rest of the reply block $(M_2[l'+1:])$, can be decrypted to retrieve $J'|biEP_{NULL}(\bar{M})|J$. The length of the Junk J', is know and can be removed. It is then possible to decode using the well known key *NULL* the fixed size message $\bar{M}$.

Note that different implementations of the EP algorithm will require different parts of the reply block to decode the relayed reply. All of this information can be extracted by modifying slightly the reply block encoding algorithm. The IGE example above, that extracts the previous plaintext and ciphertext blocks, necessary to decode an IGE message, can serve as an example.

7. IMPLEMENTATION DETAILS

Since different RSA key pairs have different moduli it is possible in some cases to identify a subset of keys which the creator of a ciphertext must be amongst. In order to avoid this we modify the RSA operation to make it key-private [3]. If the RSA ciphertext is greater than 2^{1023}, we choose new session keys, for example the sequence of hashes of the session keys $k'_i = H(k_i)$, until the ciphertext is smaller. This assures that the encryption could be the product of any 1024 bit RSA key. To avoid having always a ciphertext with a leading zero, we substitute it with a random bit[2]. If we are building a reply block the leading random bit must be a function of the master key so that the reply block can be reconstructed[3]. Of course the leading bit is set back to zero before the RSA decryption operation is applied.

In order for the last mix to apply biEP to the message or the remaining of the reply block, the boundaries must be known. For this reason messages must be of fixed length l, and the unused tail of the packet must be filled with random noise. When the last mix decodes the last header and encounters the "Final" label, it simply applies the biEP cipher to the message of this fixed size. A special label "Reply" should be included if the reply block information is included in the header. In this case the last mix decodes a smaller portion of the remaining message to find out where it should be sent.

A secure implementation of reply blocks must provide the creator with the ability to recognise the pseudonym, if any, associated with a particular reply block. For this reason Alice's Nym name is included in the reply block and must be provided to a higher layer application along with the message. A failure to do so opens a route to the *Who am I?* attack described in [9]. In this attack an adversary routes replies for one nym using the reply blocks of another to confirm that they belong to the same principal.

Finally, it is worth noting that the sender anonymous channel and the reply blocks can be combined to offer a bi-directional anonymous channel. In this scenario an anonymous sender Alice can send messages to an anonymous receiver Bob using his reply blocks. The message from Alice is appended to Bob's reply block, and then the result is encoded as a message to be sent to a random mix, with a slight modification. Instead of indicating that the chosen mix is the last one (by setting the address as 'Final'), the directory index of the first mix of Bob's reply block is included. Therefore the message gets decoded as it travels to the chosen mix, then the message gets encoded again as it travels using Bob's reply block towards Bob. Finally Bob can decode the message as if it were simply routed using his reply block.

8. SECURITY EVALUATION

In this section we evaluate Minx against the security requirements we set. We also highlight the differences with mixminion and other mix designs.

8.1 Bitwise unlinkability

It should be impossible to link the encoded and decoded versions of a packet without knowing the RSA public key or the session key used to encode it. Because of the RSA properties, and the key privacy mechanisms described above, the first 128 bytes of all messages are indistinguishable from random noise, if the private key is not known. The rest of the message is encoded using the EP mode of operation and should also be indistinguishable from noise if the session key is not known. It is also impossible to link the plaintext with the ciphertext after EP encryption (each block is encrypted with a different and unknown session key). Since the session key is never revealed, and is impossible to extract, it is not possible to use the decoded packet and link it to the RSA ciphertext. It is obvious that the random padding appended

[2]As we will see some of our security properties rely on ciphertexts and plaintexts being indistinguishable from random bit strings, to protect against malicious nodes taking advantage of tagging attacks.

[3]This leading bit influences the encoding and decoding of all subsequent blocks because of the EP error propagation properties.

to the message cannot be used to link it to any encoded input.

Replay attacks are prevented by requiring each mix to store a hash of the session key of each decoded message (we call this the message identity). The one way property of the secure hash function ensures that this record cannot be used to derive any information about the session keys, which could be used to link packets[4]. Messages containing previously seen session keys are silently dropped. Note that it is not possible to modify a message so that its identity changes without destroying its contents. The EP forward error propagation will ensure that after an honest mix has processed the packet the message is unrecoverable. Since replies are indistinguishable, and for other obvious security reasons, each reply block can only be used to relay a single message.

8.2 Information leakage

No information is leaked to the intermediate nodes or an observer. In particular the number of hops is never known to anyone. Even the final mix does not know how many hops the message has traversed: since the message is of fixed size, it will always contain the same amount of trailing noise, which gives no information about the actual route length.

Furthermore a node only knows its position if it is the last in the route or the last node of a reply block. This is necessary since a special decoding step, using biEP, has to be performed, and the message should be routed to its ultimate destination using a transport protocol. On the other hand other intermediate nodes do not know anything about their position in the path of the message, and cannot infer any information about it[5]. Given this level of protection some of the attacks described in [6], using position information, should not work.

In some cases, the final node can be the actual message destination, providing further protection against attack.

8.3 Indistinguishable replies

The routing mechanism for messages using reply blocks is exactly the same as for sender anonymous messages. Therefore a mix cannot know which of the messages it is processing are replies. This increases the anonymity sets for both categories of messages. The last mix on the path must know which messages are to be forwarded to a final destination and which are replies to be sent to the anonymous receiver. For this reason a special flag is set to "Final" or "Reply" for the appropriate processing to take place.

8.4 Simple tagging attacks

An attacker might try to modify the input, encoded message, hoping to recognise the result of the modifications in one of the decoding outputs to link them. Such attacks should not work against Minx.

Unlike Mixmaster and, to some extent, Mixminion, which use integrity checks, Minx defends against tagging attacks exclusively by making the payload extremely fragile with respect to any modification of the packet. When an adversary modifies the packet in order to embed a tag, the EP mode of operation, which is part of the decoding applied to the packet by an honest mix, propagates this change towards the end of the message. Therefore any useful information contained in the payload, since it is placed at the end of the mix packet, is destroyed.

Furthermore Minx ensures that tagging can only be detected by the very last node, and no other on the path. This is achieved by making a correctly decoded packet indistinguishable from the random strings that would be the results of tagging. Therefore an adversary gains very little information by tagging any of the message headers: not only is all the payload information lost forever, but the message follows a random path through the network until it is eventually dropped.

To summarise the defences against tagging attacks:

- If the *attacker modifies the RSA encrypted header* the resulting decryption will provide a random directory index and session key. As a result the decoded payload will be unpredictable and random, which would destroy any information useful to the adversary.
- An attacker can try to *tag the body of the message* that is encoded using the EP mode of operation. If this modification touches a header, it will not be observable, since the headers of messages cannot be distinguished from random, even by the honest mixes that decode them. As a side effect the message will become undecipherable, since the errors propagate towards the end of the packet. A modification of the payload itself would render it undecipherable, because of the biEP encoding.
- Finally a *modification of the random noise* at the tail of the packet would not yield any usable information.

We have already noted that a modified header will decode, without producing any errors, to a random next address, a random session key and a partially random payload (because of the EP forward error propagation the rest of the message will also become indistinguishable from random noise, after an honest mix has decoded it). Therefore the mix will process the message, and send it, despite it not containing any meaningful information. This prevents corrupt mixes being able to detect tagging attacks, and introduces noise into the system when tagging occurs. To ensure that these packets will not travel indefinitely we set aside a set of indexes out of the directory to encode the "Final" label, which indicates that the message should be decoded and routed to its final destination. With probability 1/8 a random number will produce a "Final" label, and the mix will process the message as final, realise that it is noise, and drop it. On average we expect such "ghost packets" to travel for 8 hops before being discarded.

9. OVERHEADS

Minx uses a hybrid crypto system that combines the asymmetric RSA cipher with the symmetric AES in EP mode. The dominant cost of encoding a packet is the number of modular exponentiations necessary, which is equal to the number of intermediate mixes the packet is to be routed

[4]We believe that because of the properties of EP it is actually safe to store session keys, since they cannot be used to decrypt partial messages, and the start of each message is obscured by RSA. Nevertheless we recommend hashes for extra security against seizure of records.

[5]The first mix might infer its position from the fact that a client node, instead of another established mix, is injecting a message.

through (on average it will be double this if the key-private RSA scheme is implemented). The client only needs to store locally a recent copy of the mix directory and a strong pass phrase to decode reply blocks.

The communications overheads that Minx introduces are limited, given the security properties sought. Assuming that 80 bit (10 bytes) keys are used, and the mix indexes in the directory are 8 bits long (1 byte), the message grows by $11n$ bytes, where n is the number of mixes it is relayed through. The fact that all messages should be of the same length, such as 32 kb, might lead to wasted bandwidth if a message to be sent is shorter. With one kilobyte set aside to contain routing information such a packet could still travel through up to 93 intermediary mixes.

On the other hand reply blocks are $11(n-1)+128$ bytes long, where n is the number of intermediate mixes the reply is to be routed through, and 128 bytes is the size of an RSA ciphertext. This is due to the fact that none of the message can be embedded in the RSA plaintext, since it is not known to the creator of the reply block: it will be appended by the user of the reply block to reply to the creator.

Finally each mix must perform a modular exponentiation for each message it decodes, and is required to store a hash of the resulting session keys (around 10 bytes). This hash might seem short but it is expected that key rotation will take place before the 2^{40} messages required to have collisions travel through the mix. There do not seem to be other attacks that can make use of the shorter digest. This list can be deleted, along with the private key of the mix, upon key rotation.

10. RELATED AND FUTURE WORK

Previous work on cryptographic mix packet formats includes [7, 19, 14]; these are vulnerable to straight forward tagging attacks [22, 21]. One mechanism invented to protect mixes against such attacks is to include integrity protection, as mixmaster [17], or the scheme analysed by Bodo Möller does [16]. Another technique is to use zero-knowledge proofs of correct shuffling to show that the inputs have not been modified, as Neff [18] does. The hybrid asymmetric and symmetric crypto technique used in Minx can be traced back to hybrid encryption with minimal length described in [20].

Minx is simply a mix packet format, and not a complete anonymous communication channel design. Minx could be used in any topology such as free or restricted routes [8, 11], or a cascade [6] topology (in which case there will be no need for the routing data present in each layer). How to generate and use dummy traffic [5] is also an orthogonal issue. Note that while any dummy traffic policy can be implemented alongside the Minx packet format, modified packets by default become dummy traffic since they turn to random noise. Finally Minx is independent of the mixing strategies that mixes implement and can be used by any of them [25].

Because of its simplicity Minx is a prime candidate for both practical use in remailer networks and route setup messages in Onion Routing [13], but also for further study. In particular we are interested in formally proving some of its properties, such as resistance to tagging attacks (Möller has done something similar for a mix packet design in [16]). A further challenge is to simplify the decoding procedure for Minx reply blocks, that at the moment requires knowledge of the reply block itself (although the mechanism we propose does not require any state to be kept per reply block). Finally it is worth exploring more primitives and constructions that provide integrity through 'fragility', such as the EP and biEP modes of operations.

11. CONCLUSIONS

Minx fulfils all the security requirements of a mix packet format and is simpler and less costly in terms of bandwidth overheads than others proposed and implemented. It allows encoding of sender anonymous messages, as well as anonymous replies, in a unified fashion that increases the anonymity provided to both types of messages. Furthermore the two mechanisms can be combined to provide a bi-directionally anonymous channel.

Minx should be secure against quite a strong attacker model: an adversary that observes the whole network, can modify packets on any link, and controls a subset of the path of each anonymous message. Despite these powers, a packet encoded using Minx does not reveal any information about its ultimate destination or contents. Novel techniques based on fragile cryptographic constructions, which destroy any useful information if modified, were used to achieve this.

Acknowledgements. This work has benefited from detailed comments and criticisms from Bodo Möller, and the anonymous reviewers' comments. George Danezis is supported by the Cambridge-MIT Institute project entitled 'The design and implementation of third generation peer-to-peer systems'.

12. REFERENCES

[1] Advanced Encryption Standard, FIPS-197. National Institute of Standards and Technology, November 2001.

[2] R. Anderson and E. Biham. Two practical and provably secure block ciphers: BEAR and LION. In *International Workshop on Fast Software Encryption*. Springer-Verlag, 1996. `http://citeseer.nj.nec.com/anderson96two.html`.

[3] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In C. Boyd, editor, *Advances in Cryptology (Asiacrypt 2001)*, volume 2248 of *LNCS*, pages 566–582, Gold Coast, Australia, 9-13 December 2001. Springer-Verlag.

[4] M. Bellare, R. Canetti, and H. Krawczyk. Message authentication using hash functions: The HMAC construction. *RSA Laboratories' CryptoBytes*, 2(1), Spring 1996.

[5] O. Berthold and H. Langos. Dummy traffic against long term intersection attacks. In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[6] O. Berthold, A. Pfitzmann, and R. Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *LNCS*, pages 30–45. Springer-Verlag, July 2000.

[7] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[8] G. Danezis. Mix-networks with restricted routes. In R. Dingledine, editor, *Privacy Enhancing Technologies workshop (PET 2003)*, volume 2760 of *LNCS*, pages 1–17, Dresden, Germany, March 2003. Springer-Verlag.

[9] G. Danezis. *Better Anonymous Communications*. PhD thesis, University of Cambridge, Computer Laboratory, 2004.

[10] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[11] R. Dingledine, V. Shmatikov, and P. Syverson. Synchronous batching: From cascades to free routes. PET 2004, 2004.

[12] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd ACM Symposium on the Theory of Computing (STOC)*, pages 542–552, 1991. Updated version at `http://citeseer.nj.nec.com/dolev00nonmalleable.html`.

[13] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding Routing Information. In R. Anderson, editor, *Proceedings of Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[14] C. Gülcü and G. Tsudik. Mixing E-mail with Babel. In *Network and Distributed Security Symposium — NDSS '96*, pages 2–16, San Diego, California, February 1996. IEEE.

[15] N. Mathewson, R. Dingledine, and G. Danezis. Type iii (mixminion) mix directory specification. Technical report, The Mixminion Project, 2004.

[16] B. Möller. Provably secure public-key encryption for length-preserving chaumian mixes. In M. Joye, editor, *Topics in Cryptology CT-RSA 2003*, volume 2612 of *LNCS*, pages 244–262, San Francisco, CA, USA, 13-17 April 2003. Springer-Verlag.

[17] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003.

[18] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In P. Samarati, editor, *ACM Conference on Computer and Communications Security (CCS 2002)*, pages 116–125. ACM Press, November 2001.

[19] C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In T. Helleseth, editor, *Advances in Cryptology (Eurocrypt '93)*, volume 765 of *LNCS*, pages 248–259, Lofthus, Norway, 23-27 May 1993. Springer-Verlag.

[20] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In W. Effelsberg, H. W. Meuer, and G. Müller, editors, *GI/ITG Conference on Communication in Distributed Systems*, volume 267 of *Informatik-Fachberichte*, pages 451–463. Springer-Verlag, February 1991.

[21] B. Pfitzmann. Breaking efficient anonymous channel. In A. D. Santis, editor, *Advances in Cryptology (Eurocrypt '94)*, volume 950 of *LNCS*, pages 332–340, Perugia, Italy, 9-12 May 1994. Springer-Verlag.

[22] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of MIXes. In *Proceedings of EUROCRYPT 1989*. Springer-Verlag, LNCS 434, 1990.

[23] J. B. Postel. Simple mail transfer protocol. Technical report, Request for comments number 821, August 1982.

[24] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[25] A. Serjantov, R. Dingledine, and P. Syverson. From a trickle to a flood: Active attacks on several mix types. In F. Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.

[26] P. D. V. Gligor. Infinite garble extension. Technical report, NIST, 10 November 2000.