Sphinx: A Compact and Provably Secure Mix Format

George Danezis
Microsoft Research
Cambridge, UK
gdane@microsoft.com

Ian Goldberg
University of Waterloo
Waterloo, ON, Canada
iang@cs.uwaterloo.ca

Abstract

Sphinx is a cryptographic message format used to relay anonymized messages within a mix network. It is more compact than any comparable scheme, and supports a full set of security features: indistinguishable replies, hiding the path length and relay position, as well as providing unlinkability for each leg of the message's journey over the network. We prove the full cryptographic security of Sphinx in the random oracle model, and we describe how it can be used as an efficient drop-in replacement in deployed remailer systems.

1. Introduction

Mix networks were proposed by David Chaum [7] in 1981 as an efficient means to achieve anonymous communications. A *mix* (or a *mix node*) is simply a message relay that accepts a batch of encrypted messages, decrypts them and sends them on their way. An observer of a mix node should be unable to link incoming and outgoing messages; this provides anonymity to the users of the network. A number of theoretical [12], [15], [9], [5], [21] as well as deployed systems [16], [8] have been proposed that further develop the idea of mixes. The Mixmaster network [16] is currently composed of about 25 reliable remailers, while the newer Mixminion [8], adding the ability to reply anonymously to messages, is composed of about 20 reliable nodes.

Anonymizing messages through a mix network comes at a cost: the messages are batched and therefore delayed, as well as padded to a standard length to prevent traffic analysis. Furthermore, multiple encryption layers have to be used to encapsulate the routing information necessary to relay the message through a sequence of mixes. The cryptographic mechanism used to deliver this routing information to each intermediate mix, as well as to transform the message as it travels through the network, is called the *cryptographic packet format*. The cryptographic mechanism is to some extent independent from other traffic analysis protections offered by the mix network, as long as they guarantee that some aspects of the routing information, such as path length or position in the path, are not leaked [4].

The minimum overhead introduced by the cryptographic packet format impacts the types of traffic that can realistically be anonymized. Previous work, like Mixminion [8] and Minx [9] added an overhead of at least a full RSA ciphertext (at least 256 additional bytes for modern 128-bit security levels). Provable designs, such as the ones proposed by Möller [15], Camenish and Lysyanskaya [5] or Shimshock *et al.* [21], use multiple RSA-sized ciphertexts to relay information for each stage of the mixing, making the header necessary for anonymization many kilobytes long. Such formats may be acceptable for relaying large email messages, but add a significant overhead to short messages, the length of Instant Messaging or SMS messages (that are up to 160 characters.) It is therefore of great importance to devise cryptographic schemes that are *compact* to efficiently anonymize those classes of traffic. Furthermore, anonymous replies rely on cryptographic addresses that are of similar size to the headers required to route messages thought the network. Compact packet formats directly lead to compact addresses and thus to cheaper receiver anonymity.

Traditionally, cryptographic packet formats have been based on heuristic security arguments. From early on it became apparent that these complex cryptographic systems are difficult to get right: the original scheme by Chaum [7] was shown to have cryptographic weaknesses by Pfitzmann and Pfitzmann [18] and Minx [9] leaked information which theoretically allowed an adversary to extract the full plaintext of messages [21]. As a result several authors proposed provably secure packet formats. Some of them only provide the bare minimum functionality [15], [5], and in particular no provision for anonymous replies, while others suffer a significant transmission overhead [21]. It has so far been an open problem to devise a compact

and provably secure packet format.

Our key contribution is *Sphinx*: a cryptographic packet format that can be used to route messages over a mix network. Sphinx provides all features expected by modern remailer applications:

- It provides *bitwise unlinkability*, making it cryptographically difficult to link incoming and outgoing messages from a mix. This is the basic property all packet formats provide.
- It allows for paths up to an arbitrary maximum length, while hiding the number of hops a message has travelled so far, as well as the actual number of mixes on the path of a message. These are hidden even from the mixes that are processing the messages.
- The processing of reply messages is indistinguishable from the processing of normal "forward" messages. The anonymity sets of both types of traffic are therefore confounded, providing greater protection for both.
- It resists any active *tagging attacks*, where the adversary modifies and re-injects messages to extract information about their destinations or content.
- It is compact: for 128-bit security, the overhead is only 32 bytes plus a single group element (not one per hop) plus 32 bytes of routing and integrity protection information per hop. The size of the group element can be as small as 32 bytes using Dan Bernstein's Curve25519 elliptic curve library [3]; the element is the x-coordinate (in $GF(2^{255}-19)$) of a point on an elliptic curve. For example, the header of a Sphinx message with a maximum path length of 5 mixes can be encoded in as little as 224 bytes.

From a systems perspective, Sphinx is designed as a drop-in replacement for the Mixminion packet format [8]. It makes the same systems and security assumptions as Mixminion, but is more compact and cryptographically provably secure. This means that Sphinx can be easily integrated with the Mixminion software to take advantage of the thousands of lines of robust client and server code. Our reference implementation of Sphinx, which provides all of the functionality needed for clients, mix nodes, and the nymserver, and which works either over a 2048-bit prime field or Curve25519, is less than 600 lines of Python code (including simple tests).

Our description of Sphinx will proceed in the following fashion: section 2 provides an overview of the threat model, requirements and design rationale of Sphinx; sections 3 and 4 provide a formal definition of the cryptographic format and the associated proofs of security respectively. In section 5 we discuss the efficiency of the scheme and compare it with other proposals. Finally we provide some concluding remarks in section 6.

2. Design overview

Mix networks achieve anonymity by relaying messages over a sequence of mixes, called the *path*. The sender cryptographically encodes a message, which is partially decoded by each mix along the path. As long as a single mix in the path is honest, meaning that it does not share its secrets with an adversary, the message will benefit from some anonymity.

2.1. Threat Model & Requirements

It is traditional to consider the security of cryptographic packet formats against an active adversary that is able to observe all traffic in the network, as well as to intercept and inject arbitrary messages. Furthermore, we assume that some, but not all, of the mixes in the path of a relayed message are corrupt; i.e., under the direct control of an adversary that knows all their keys, and other secrets, and is able to fully control their functioning. The principal aim of the adversary is to extract some information about the ultimate destination of mixed messages through inferring the final address or some of their contents. The security properties of cryptographic mix packet formats prevent or tightly control any such information leakage.

Specifically, Sphinx needs to ensure that if multiple messages enter an honest mix and are batched together, it is not feasible to link an output message to any input message with probability significantly higher than uniform. This is the fundamental requirement of any packet format. Yet modern packet formats also control the amount of information leaked to mixes (including dishonest mixes) along the path. A dishonest mix should not be able to infer either the full length of the path of a particular message, or its own position on that path. Simple traffic analysis detects if a mix is first or last on the path, so we do not consider this a compromise.

An advantage of mix networks is that they provide a unified mechanism for both sender and receiver anonymity. Alice may encode a *single-use reply block* [8] and attach it to an anonymous message destined to Bob. Bob cannot know that the originator of the message is Alice, but can use the reply block contained in the message as an address to send a reply. The reply is then routed through the network until

it reaches Alice. In both cases Alice benefits from anonymity properties, first as the sender of the message (sender anonymity) and the second time as the anonymous receiver of a message (receiver anonymity). Nymservers [14] have been developed as services that make use of anonymous replies to bridge the world of mix networks with traditional email. Users can send normal email to pseudonymous email addresses, which are then routed through the mix network using anonymous reply blocks. We note that the network does *not* attempt to keep it a secret that Bob is talking to some particular pseudonym; we assume Bob's email communication is likely to be unencrypted (though orthogonal end-to-end encryption mechanisms can of course also be used). What it *does* protect is the fact that that pseudonym belongs to Alice.

To increase the security of both sender- and receiver-anonymous messages, the two kinds of messages should use the same network and the same relay mechanisms. Hence it is a requirement of the cryptographic packet format that forward and reply messages be cryptographically indistinguishable; this makes the task of engineering packet formats more difficult. Traditional methods for achieving non-malleability (such as providing a MAC over the full header and body of the relayed message) are not readily available, as the reply message body is unknown at the time the reply block is created.

Finally, Sphinx is required to be resistant to active attacks: an adversary can use any corrupt node to inject arbitrary messages into honest nodes in an attempt to extract information. Sphinx protects against replay attacks—where the adversary reinjects a previously seen message verbatim—and tagging attacks—where the adversary modifies a message before reinjecting it into the network. In neither event will Sphinx allow the adversary to learn any information about the final destination of the message or its contents. Other denial-of-service and flooding attacks ($n-1$ attacks) [20] are not considered in the threat model, since they are not cryptographic in nature and are handled by orthogonal mechanisms [12], [13]. Similarly, active traffic analysis attacks, such as those based on dropping messages or flooding links and nodes to perform remote network measurements, cannot be fixed by the cryptographic format alone, and must be dealt with by the high-level mix strategies.

2.2. Design Rationale

Sphinx is based on the idea that a mix packet format should encapsulate enough information to cryptographically secure a confidential and integrity-protected channel to each of the mixes on a message's path. This requires keys to be shared, or distributed, securely to each of the mixes on the path, in order that they may decode the routing information, as well as other parts of the message. Traditionally, this has been done with RSA [19] encryption, while Sphinx instead uses Diffie-Hellman [10].

At the heart of the Sphinx key distribution strategy lies a single element of a cyclic group of prime order satisfying the decisional Diffie-Hellman assumption. This element is used by each mix on the path to derive a secret that is shared with the original sender of the message—a set of keys that can be used for encryption, integrity protection, etc. are further extracted from this shared secret. The element used for key derivation cannot be transported unaltered throughout the path, however, as this would lead to linkable messages. To avoid this, the element is blinded at each mixing step to make it indistinguishable from any other output element. The blinding factors are extracted from the shared secrets, and so both senders and mixes can perform all operations necessary to extract keys used at each hop of the mix message processing. There are many possible choices for the cyclic group; two common ones are a subgroup of the multiplicative group of a prime field, and an elliptic curve group. The latter in particular leads to a very compact design, since for 128-bit security, group elements can be expressed in just 32 bytes, as opposed to 256–384 bytes for a prime field of similar strength.

Besides extracting the shared key, each mix has to be provided with authentic and confidential routing information to direct the message to the subsequent mix, or to its final destination. We achieve this by a simple encrypt-then-MAC mechanism. A secure stream cipher or AES in counter mode is used for encryption, and a secure MAC (with some strong but standard properties) is used to ensure no part of the message header containing routing information has been modified. Some padding has to be added at each mix stage, in order to keep the length of the message invariant at each hop.

The steps involved in decoding and routing the message at each mix are rather simple. Their full technical description is provided in section 3.6 and is illustrated in the corresponding Figure 3. In summary:

1) The mix receives the message and, using the element from the cyclic group and its private key, extracts a set of shared session keys.
2) The MAC of the message is checked to ensure that the header has not been modified.
3) Some padding (of all zeros) is added at the end of the message to keep the length invariant.

4) The header of the message is decrypted (including the newly added padding), the element blinded and the payload of the message decrypted.
5) The routing information and next MAC are extracted from the decrypted header, and the resulting message is forwarded to the next destination.

Senders encode a message by deriving all session keys, wrapping the message in multiple layers of encryption, and calculating the correct message authentication codes for each stage of the journey. Calculating the correct MACs is not a trivial task: the successive layers of padding that are encrypted at each stage of the mixing have to be included in the integrity check. The MACs ensure that a modified header is detected immediately.

The payload of the message is kept separate from the mix header used to perform the routing. It is decrypted at each stage of mixing using a block cipher with a large block size (the size of the entire message), such as LIONESS [1]. In case the adversary modifies the payload in transit, any information contained in it becomes irrecoverable. Sender-anonymous messages contain the final address of the message, as well as the message itself as part of the payload, and so any modification destroys this information.

Anonymous replies are equally simple to construct: the intended receiver of the reply (who will benefit from the anonymity properties) builds a mix header addressed back to herself with no payload. This header acts as an anonymous reply address, and can be included in a message to give anyone the ability to reply. Some additional information, such as the address of the first mix hop, is also needed.

A reply is built by attaching a message to the reply address and routing it through the mix network. The processing of the reply message is identical to the processing of forward messages, leading to simplicity of implementation and larger anonymity sets.

3. Formal protocol description

3.1. Notation

Let κ be a security parameter. An adversary will have to do about 2^κ work to break the security of Sphinx with nonnegligible probability. We suggest using $\kappa = 128$.

Let r be the maximum number of nodes that a Sphinx mix message will traverse before being delivered to its destination. We suggest $r = 5$.

Define the following:

$\mathcal{G}$: A prime-order cyclic group satisfying the Decisional Diffie-Hellman Assumption. $\mathcal{G}^*$ is the set of non-identity elements of $\mathcal{G}$. The element g is a generator of $\mathcal{G}$, and q is the (prime) order of $\mathcal{G}$, with $q \approx 2^{2\kappa}$.

A number of hash functions, which we model by random oracles:

- $h_\mu : \mathcal{G}^* \to \{0,1\}^\kappa$, used to key μ, below
- $h_\rho : \mathcal{G}^* \to \{0,1\}^\kappa$, used to key ρ, below
- $h_\pi : \mathcal{G}^* \to \{0,1\}^\kappa$, used to key π, below
- $h_\tau : \mathcal{G}^* \to \{0,1\}^{2\kappa}$, used to identify previously seen elements of $\mathcal{G}^*$
- $h_b : \mathcal{G}^* \times \mathcal{G}^* \to \mathbb{Z}_q^*$, used to compute blinding factors

We implement these functions with appropriately truncated SHA-256 hash functions.

$\mu : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$: a Message Authentication Code (MAC). We normally model μ as a pseudo-random function (PRF). However, in one part of the proof (section 4.2), the adversary gets to know the key to the MAC. In this case, simply being a PRF guarantees nothing, whereas we still want μ with a known key to behave like a hash function. For this reason, we model μ as a random oracle in that section (which of course is stronger than a PRF). In a realistic implementation, we would use a MAC based on a hash function, such as SHA256-HMAC-128.

$\rho : \{0,1\}^\kappa \to \{0,1\}^{(2r+3)\kappa}$: a pseudo-random generator (PRG). A PRG is the basis for any stream cipher: the key is fed as an input to the PRG, which outputs a long pseudorandom string. This string is XORed with the plaintext to yield the ciphertext, or with the ciphertext to recover the plaintext. As above, in section 4.2 the adversary will be able to know the input to the PRG, which removes all of the PRG's security properties. So again, for that section, we model ρ as a random oracle. We can implement ρ with any secure stream cipher, or any secure block cipher in counter mode, which operates in the same way.

$\pi : \{0,1\}^\kappa \times \{0,1\}^{\ell_\pi} \to \{0,1\}^{\ell_\pi}$: a family of pseudo-random permutations (PRPs). ℓ_π will be the size of the message bodies that can be transmitted over Sphinx (plus κ bits of overhead). Given any $(k,x) \in \{0,1\}^\kappa \times \{0,1\}^{\ell_\pi}$, both $\pi(k,x)$ and $\pi^{-1}(k,x)$ should be easy to compute. (The latter is the unique value $y \in \{0,1\}^{\ell_\pi}$ such that $\pi(k,y) = x$.) π^{-1} should also be a family of pseudo-random permutations. We use the LIONESS [1] PRP to implement π.

$\mathcal{N} \subset \{0,1\}^\kappa$: a set of mix node identifiers. Each node $n \in \mathcal{N}$ has a private key $x_n \in_R \mathbb{Z}_q^*$ and a public key $y_n = g^{x_n} \in \mathcal{G}^*$. We assume the presence of a PKI that publishes an authenticated list of all (n, y_n) pairs.

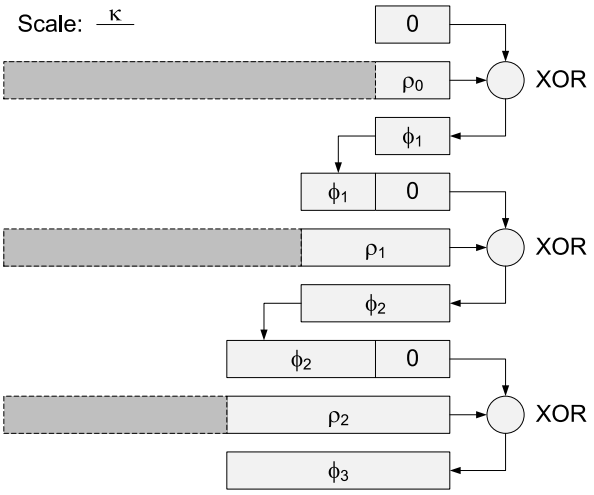

Figure 1. Construction of the filler strings ϕ_i for $\nu = r = 4$. Here, ρ_i is the final $2(i+1)\kappa$ bits of $\rho(h_\rho(s_i))$.

$\mathcal{D} \subset \{0,1\}^{\leq 2r\kappa}$: a set of destination addresses, usually normal email addresses. It must be the case that $\mathcal{N} \cap \mathcal{D} = \emptyset$ and that $\mathcal{N} \cup \mathcal{D}$ is prefix-free. Note that $\mathcal{N} \cap \mathcal{D} = \emptyset$ does not imply that end users of Sphinx cannot themselves run Sphinx nodes; it is just that the identifier for their node (in $\mathcal{N}$) will be different from their email address. One of the elements $* \in \mathcal{D}$ is distinguished.

The notation 0_a means the string of 0 bits of length a, $x_{[a..b]}$ means the substring of x consisting of bits a through b, inclusive (the leftmost bit of x is bit 0), $\|$ denotes concatenation, $|s|$ is the length of string s, and ϵ is the empty string.

3.2. Creating a mix header

This section describes the procedure to create a Sphinx mix message header. It is used as a subroutine for the procedures to create forward messages and single-use reply blocks in sections 3.3 and 3.4, respectively.

Input: a destination address $\Delta \in \mathcal{D}$, an identifier $I \in \{0,1\}^\kappa$ and a sequence of mix nodes $\{n_0, n_1, \ldots, n_{\nu-1}\}$ with $\nu \leq r$. It must also be the case that $|\Delta| \leq (2(r-\nu)+2)\kappa$.

Pick a random $x \in_R \mathbb{Z}_q^*$.

Compute a sequence of ν tuples $(\alpha_0, s_0, b_0), \ldots, (\alpha_{\nu-1}, s_{\nu-1}, b_{\nu-1})$ as follows:

- $\alpha_0 = g^x$, $s_0 = y_{n_0}^x$, $b_0 = h_b(\alpha_0, s_0)$
- $\alpha_1 = g^{xb_0}$, $s_1 = y_{n_1}^{xb_0}$, $b_1 = h_b(\alpha_1, s_1)$
- $\ldots$

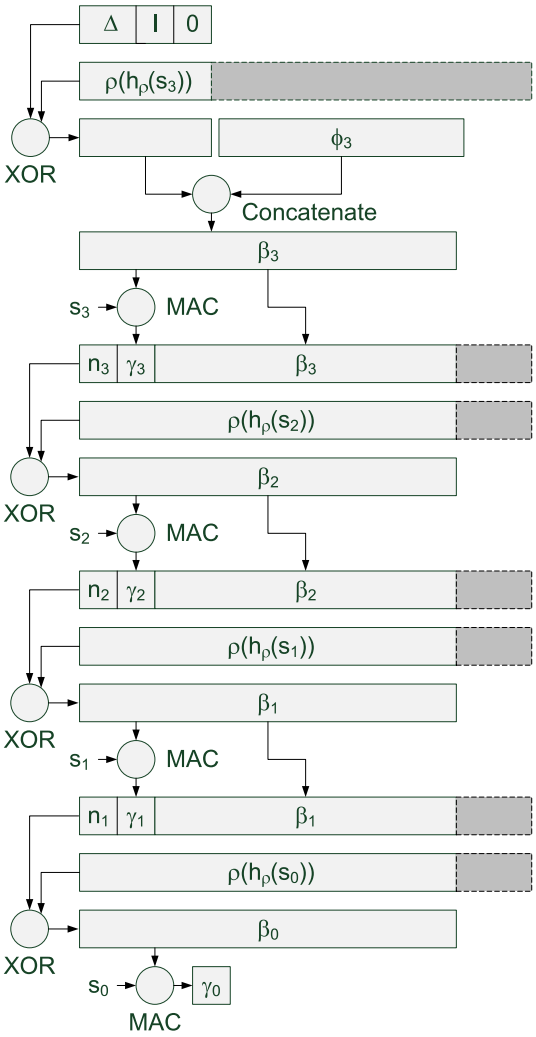

Figure 2. Construction of (β_0, γ_0) for a Sphinx mix header, with $\nu = r = 4$. The construction of ϕ_3 from Figure 1 ensures that the truncated part of β_{i+1} equals the truncated part of $\rho(h_\rho(s_i))$, indicated by the dotted lines and shading, for each $0 \leq i < \nu$.

- $\alpha_{\nu-1} = g^{xb_0 b_1 \cdots b_{\nu-2}}$, $s_{\nu-1} = y_{n_{\nu-1}}^{xb_0 b_1 \cdots b_{\nu-2}}$, $b_{\nu-1} = h_b(\alpha_{\nu-1}, s_{\nu-1})$

The α_i are the group elements, the s_i are the Diffie-Hellman shared secrets, and the b_i are the blinding factors.

Compute the filler strings $\phi_0, \ldots, \phi_{\nu-1}$:

- $\phi_0 = \epsilon$
- For $0 < i < \nu$, $\phi_i = \{\phi_{i-1} \| 0_{2\kappa}\} \oplus \left\{\rho(h_\rho(s_{i-1}))_{[(2(r-i)+3)\kappa..(2r+3)\kappa-1]}\right\}$

Note that $|\phi_i| = 2i\kappa$. This step is illustrated in Figure 1.

Compute a sequence of mix headers

$M_{\nu-1}, M_{\nu-2}, \ldots, M_0$ as follows: $M_i = (\alpha_i, \beta_i, \gamma_i) \in \mathcal{G}^* \times \{0,1\}^{(2r+1)\kappa} \times \{0,1\}^\kappa$ where:

- $\beta_{\nu-1} = \left\{\left\{\Delta \| I \| 0_{(2(r-\nu)+2)\kappa - |\Delta|}\right\} \oplus \left\{\rho(h_\rho(s_{\nu-1}))_{[0..(2(r-\nu)+3)\kappa-1]}\right\}\right\} \| \phi_{\nu-1}$
- $\beta_i = \left\{n_{i+1} \| \gamma_{i+1} \| \beta_{i+1[0..(2r-1)\kappa-1]}\right\} \oplus \rho(h_\rho(s_i))_{[0..(2r+1)\kappa-1]}$ for $0 \le i < \nu - 1$
- $\gamma_i = \mu(h_\mu(s_i), \beta_i)$ for $0 \le i \le \nu - 1$

The above step is illustrated in Figure 2.

Output: the mix header M_0 and the sequence of shared secrets $s_0, \ldots, s_{\nu-1}$.

3.3. Creating a forward message

This section gives the procedure used to create a forward message to be sent through the Sphinx network.

Input: a message m, a destination address Δ and a sequence of mix nodes $\{n_0, n_1, \ldots, n_{\nu-1}\}$ with $\nu \le r$.

Compute the mix header M_0 and the sequence of shared secrets $s_0, \ldots, s_{\nu-1}$ as above, passing the distinguished element $* \in \mathcal{D}$ as the destination address and 0_κ as I. Compute:

- $\delta_{\nu-1} = \pi(h_\pi(s_{\nu-1}), 0_\kappa \| \Delta \| m)$
- $\delta_i = \pi(h_\pi(s_i), \delta_{i+1})$ for $i = \nu - 2, \ldots, 0$

Output: the pair (M_0, δ_0)

The forward message is this pair (M_0, δ_0), and should be sent to n_0.

3.4. Creating a single-use reply block

This procedure is used to create a single-use reply block.

Input: a destination address Δ and a sequence of mix nodes $\{n_0, n_1, \ldots, n_{\nu-1}\}$ with $\nu \le r$. Δ should be the user's own address.

Pick a random identifier $I \in_R \{0,1\}^\kappa$ and compute the mix header M_0 and the sequence of shared secrets $s_0, \ldots, s_{\nu-1}$ as above.

Pick a random key $\tilde{k} \in_R \{0,1\}^\kappa$.

Output: I, the tuple $(\tilde{k}, h_\pi(s_0), \ldots, h_\pi(s_{\nu-1}))$, and the tuple $(n_0, M_0, \tilde{k})$.

Store the tuple $(\tilde{k}, h_\pi(s_0), \ldots, h_\pi(s_{\nu-1}))$ in a local table indexed by I. Send $(n_0, M_0, \tilde{k})$ to the nymserver over a secure channel, to be indexed under the user's pseudonym. This can be done, for example, by encrypting it with the nymserver's public key, signing it with the pseudonym's private key, and sending the message to the nymserver using the Sphinx forward channel.

3.5. Using a single-use reply block

When the nymserver receives a message m destined for a pseudonym, it will look up a previously unused $(n_0, M_0, \tilde{k})$ tuple indexed by that pseudonym. It will then send $(M_0, \pi(\tilde{k}, 0_\kappa \| m))$ to n_0 and remove the tuple from its index.

3.6. Message processing by mix nodes

Messages received by mix nodes are of the form $(M, \delta) = ((\alpha, \beta, \gamma), \delta) \in \mathcal{G}^* \times \{0,1\}^{(2r+1)\kappa} \times \{0,1\}^\kappa \times \{0,1\}^{\ell_\pi}$. (The node should ensure that the message is in this form; in particular, that $\alpha \in \mathcal{G}^*$.) When mix node n, with private key x_n, receives such a message, it proceeds as follows:

Compute the shared secret $s = \alpha^{x_n}$. If $h_\tau(s)$ is already in this node's table of seen message tags, discard the message. (Note that this table can be flushed whenever the node rotates its private key.) Otherwise, continue by comparing γ to $\mu(h_\mu(s), \beta)$. If they do not match, discard the message. Otherwise, store $h_\tau(s)$ in the table of seen message tag, and continue by decrypting the suitably padded β (as a stream cipher, XORing the output of the PRG ρ) to get $B = \{\beta \| 0_{2\kappa}\} \oplus \rho(h_\rho(s))$.

Use the prefix-freeness of $\mathcal{N} \cup \mathcal{D}$ to uniquely parse a prefix of B as $n \in \mathcal{N} \cup \mathcal{D}$. (If this is not possible, the message is discarded.)

If $n \in \mathcal{N}$ is found: This message is destined for another Sphinx node. Compute the blinding factor $b = h_b(\alpha, s)$, and let $\alpha' = \alpha^b$. Let $\gamma' = B_{[\kappa..2\kappa-1]}$, $\beta' = B_{[2\kappa..(2r+3)\kappa-1]}$, and $\delta' = \pi^{-1}(h_\pi(s), \delta)$. Send $((\alpha', \beta', \gamma'), \delta')$ to n. Figure 3 illustrates the processing steps involved in this case, as an example of how the decoding process works.

If $n = *$ is found: The current node is the exit node for a forward message. Let $\delta' = \pi^{-1}(h_\pi(s), \delta)$. If ${\delta'}_{[0..\kappa-1]} = 0_\kappa$, parse ${\delta'}_{[\kappa..\ell_\pi-1]}$ as $\Delta \| m$ for $\Delta \in \mathcal{D}$ using the prefix-freeness of $\mathcal{D}$. If this is successful, m should be a plaintext message, and is sent to Δ. Otherwise, the message has been tampered with and is discarded.

Otherwise, if $n \in \mathcal{D} \backslash \{*\}$ is found: The current node is the exit node for a reply message, and n is the owner of a pseudonym. Let $I = B_{[|n|..|n|+\kappa-1]}$ and $\delta' = \pi^{-1}(h_\pi(s), \delta)$. Send (I, δ') to n.

3.7. Reply message processing by pseudonym owners

Upon receiving (I, δ), a pseudonym owner looks up (and subsequently removes) $(\tilde{k}, k_0, \ldots, k_{\nu-1})$

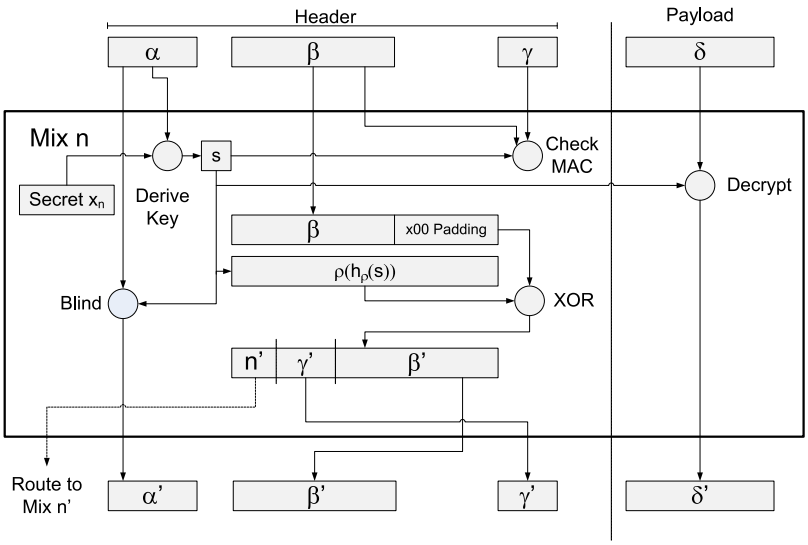

Figure 3. The processing of a Sphinx message $((\alpha, \beta, \gamma), \delta)$ into $((\alpha', \beta', \gamma'), \delta')$ at Mix n.

in its table indexed by I, and computes $\delta' = \pi^{-1}\left(\tilde{k}, \pi\left(k_0, \pi\left(k_1, \cdots \pi\left(k_{\nu-1}, \delta\right)\cdots\right)\right)\right)$ and $m = \delta'_{[\kappa..\ell_\pi - 1]}$. If $\delta'_{[0..\kappa-1]} = 0_\kappa$ then accept m as the received message.

4. Proof of security

From a cryptographic point of view, mix protocols like Sphinx share many properties with onion routing protocols. At a minimum, we desire our mix protocol to have all of the security properties of onion routing.

In [5], Camenisch and Lysyanskaya give four properties of an onion routing protocol: correctness, integrity, wrap-resistence, and security, all detailed below. They show that any onion routing protocol having all of these properties realizes ideal onion routing functionality in the Universal Composability model [6]. This means that an adversary against a protocol with these four properties has no better chance of success than an adversary against an *ideal* protocol; that is, at a high level, one in which adversaries (even ones that control some of the mix nodes) have no access to the underlying cryptographic implementation, but rather can observe only opaque identifiers for messages.

For our mix network, we would like, in addition to the above properties, that adversaries in the middle of a path should be unable to distinguish forward messages from replies (unlike the situation in [5]). It is clear that adversary nodes at the edges of the network—that is, nodes that deliver messages to users who are not themselves nodes—are necessarily able to distinguish forward from reply messages: outgoing forward messages are in plaintext, since messages should be deliverable to arbitrary parties on the Internet who have no special software installed; on the other hand, outgoing reply messages to the pseudonym owner are encrypted. Entry nodes also receive forward messages from arbitrary end users, but receive reply messages from the nymserver (the forward and reply messages are *cryptographically* indistinguishable, however). But between the entry and the exit, nodes should be unable to distinguish the two cases.

Formally, under the assumptions on the components given in section 3.1, Sphinx realizes ideal onion routing functionality in the Universal Composability model (as defined in [5]), and also makes forward and reply messages indistinguishable to middle mix nodes. We prove this result in the following four sections.

4.1. Correctness

It is straightforward by inspection that the protocol works correctly in the absence of an adversary; that is, it processes the mix messages correctly, sends the right intermediate mix messages to the right mixes, and finally sends the right message to the right destination.

4.2. Integrity

The second requirement of [5] is that an adversary cannot construct a mix message that will travel through a path of more than N honest nodes, for some fixed bound N, except with negligible probability. We show that Sphinx satisfies this requirement, with $N = r + 1$, even if we allow the adversary to know all private keys x_n in the system. (Note that the adversary knows the

nodes' private keys, but the nodes still behave honestly, according to the protocol.) This last adversarial power is what necessitates modelling μ and ρ as something stronger than the usual PRF and PRG notions. Again, for the purposes of this section, we treat them as random oracles.

Note that although the Sphinx protocol specifies that no more than r node identifiers get embedded into a Sphinx header, it is in fact possible to embed up to $r+1$ such identifiers, as long as the embedded $\Delta = *$ and is very short (less than κ bits). This means an adversary can indeed construct a mix message that will have path length $r+1$. This is not a problem for our proof, since the proof only requires that there is *some* upper bound on the path length. We show that an adversary cannot construct a mix message that results in a path length *greater than* $N = r + 1$, and that is sufficient.

We assume the adversary does significantly less than 2^κ work, and show that the probability of producing a requisite mix message is negligible.

Let a mix message constructed by the adversary be $((\alpha_0, \beta_0, \gamma_0), \delta_0)$, and sent to node n_0. That node processes it to produce $((\alpha_1, \beta_1, \gamma_1), \delta_1)$, which is sent to n_1, etc.

Node n_i will successfully process a message and send it on to the next node if and only if the following all hold:

- n_i has never before (during the life of its current private key x_{n_i}) processed a mix message with the same α_i (since the map $\alpha_i \mapsto \alpha_i^{x_{n_i}}$ is bijective and h_τ is collision-resistant)
- $\gamma_i = \mu\left(h_\mu\left(\alpha_i^{x_{n_i}}\right), \beta_i\right)$
- there is a prefix of $B_i = \{\beta_i \| 0_{2\kappa}\} \oplus \rho(h_\rho(\alpha_i^{x_{n_i}}))$ which is in $\mathcal{N} \cup \mathcal{D}$; this will be n_{i+1} if it is in $\mathcal{N}$.

If these hold, then the first κ bits of B_i will be n_{i+1} itself, the next κ bits will be γ_{i+1}, and the remaining $(2r+1)\kappa$ bits will be β_{i+1}. Note that $\beta_{i+1} = \left\{\beta_{i[2\kappa..(2r+1)\kappa-1]} \oplus \rho(h_\rho(\alpha_i^{x_{n_i}}))_{[2\kappa..(2r+1)\kappa-1]}\right\} \| \left\{\rho(h_\rho(\alpha_i^{x_{n_i}}))_{[(2r+1)\kappa..(2r+3)\kappa-1]}\right\}$. In particular, the leftmost 2κ bits of β_i are used to construct n_{i+1} and the MAC γ_{i+1}; the remaining $(2r-1)\kappa$ bits of β_i are shifted left to form (after decryption by XORing with a substring of an output of ρ) the leftmost $(2r-1)\kappa$ bits of β_{i+1}; the rightmost 2κ bits of β_{i+1} are simply a substring of an output of ρ.

Consider the following problem P: Let $f_0, f_1, \ldots, f_{2^\kappa-1}$ be a family of random oracles with range $\{0,1\}^\kappa$. Let $\hat{\rho}$ and ρ_0 be other random oracles with range $\{0,1\}^\kappa$. (The domains do not matter.) The problem is to find x and y such that $\hat{\rho}(x) = f_{\rho_0(x)}(y)$. We claim that an adversary has only a negligible chance of solving problem P if he performs significantly less than 2^κ work.

Proof: For each $i \in \{0,1\}^m$, let r_i be the number of times the adversary called ρ_0 and had it output i, and let F_i be the number of times he called f_i. (We can assume the adversary never calls $\hat{\rho}$ on an input unless he also calls ρ_0 on that input.) Then the adversary has $\sum_i r_i F_i$ chances to solve problem P, and each chance is successful with probability $2^{-\kappa}$. Let A be the total number of calls the adversary makes to ρ_0, and let i^* be the most common output, occurring A^* of the A times. To maximize $\sum_i r_i F_i$ while holding $\sum_i (r_i + F_i)$ constant, the adversary should only query f_{i^*}, and not any of the other f_i. Suppose he does so B times. Then his probability of success is bounded above by $\frac{A^* \cdot B}{2^\kappa}$, having done $A + B$ work.

If $A = B \approx 2^{\kappa\left(1-\frac{1}{w}\right)}$ for some w, then we expect to find w-collisions in the A outputs of ρ_0, but not $w+1$-collisions [17], so we expect $A^* = w$. Then the success probability is bounded above by $\frac{A^* \cdot B}{2^\kappa} = w \cdot 2^{-\frac{\kappa}{w}}$. If $A = B \approx 2^m$, then $w = \frac{\kappa}{\kappa - m}$, and this probability bound is $\frac{\kappa}{\kappa-m} \cdot 2^{m-\kappa}$, as required.

Note the difference between this problem and a standard collision problem (such as $\hat{\rho}(x) = f(y)$), which would have success probability $2^{2m-\kappa}$ for doing 2^m work, and to a standard search problem (such as $\hat{\rho}(x) = f(x,y)$), which would have success probability $2^{m-\kappa}$ for doing 2^m work. □

Now suppose an adversary can construct a mix message $((\alpha_0, \beta_0, \gamma_0), \delta_0)$ which is successfully processed by mix nodes n_0, n_1, …, n_N, whose private keys x_i are known to the adversary. We will show that this means that the adversary can solve the above problem P.

Given such a message, the adversary can process it in the manner of each mix node in turn, to generate $((\alpha_1, \beta_1, \gamma_1), \delta_1)$, $((\alpha_2, \beta_2, \gamma_2), \delta_2)$, …, $((\alpha_N, \beta_N, \gamma_N), \delta_N)$. Now since node n_N successfully processes this last message, it must be the case that $\gamma_N = \mu(h_\mu(\alpha_N^{x_N}), \beta_N)$.

For notational convenience, for $0 \le i \le r$, define $\rho_i(x)$ to be $\rho(x)_{[(2(r-i)+2)\kappa..(2r+3)\kappa-1]} \| 0_{2(r-i)\kappa}$; that is, the last $(2i+1)\kappa$ bits of $\rho(x)$, followed by $2(r-i)\kappa$ bits of zeros. Also define $\hat{\rho}_i(x)$ to be $\rho(x)_{[(2(r-i)+1)\kappa..(2(r-i)+2)\kappa-1]}$; that is, the block of κ bits of $\rho(x)$ immediately preceding the bits selected for $\rho_i(x)$. Finally, for $1 \le i \le r+1$, define $\tilde{\rho}_i(x)$ to be $\rho(x)_{[(2i-1)\kappa..(2i+1)\kappa-1]}$.

A careful, but straightforward, calculation shows that $\gamma_N = a_0 \oplus a_1 \oplus \cdots \oplus a_{N-1}$, where $a_j = \hat{\rho}_j(h_\rho(\alpha_j^{x_j}))$.

Similarly, $\beta_N = b_0 \oplus b_1 \oplus \cdots \oplus b_{N-1}$, where $b_j = \rho_j(h_\rho(\alpha_j^{x_j}))$.

Now consider the function $g(p_0, k_1, k_2, \ldots, k_{N-1}, k_\mu) = \mu\left(k_\mu, (p_0 \| 0_{2r\kappa}) \oplus \left(\bigoplus_{i=1}^{N-1} \rho_i(k_i)\right)\right)$. We claim that the adversary who does less than 2^κ work cannot distinguish this g from a truly random function which takes the same inputs. Why is this? Since μ is a random oracle, the only way the adversary could distinguish the situations is if it could generate a pair of inputs $(p_0, k_1, k_2, \ldots, k_{N-1}, k_\mu)$ and $(p'_0, k'_1, k'_2, \ldots, k'_{N-1}, k'_\mu)$ which cause the corresponding arguments of the call to μ to be equal. Clearly we must have $k_\mu = k'_\mu$.

Let $B = (p_0 \| 0_{2r\kappa}) \oplus \left(\bigoplus_{i=1}^{N-1} \rho_i(k_i)\right)$ and $B' = (p'_0 \| 0_{2r\kappa}) \oplus \left(\bigoplus_{i=1}^{N-1} \rho_i(k'_i)\right)$. Note that the last 2κ bits of B are just $\tilde{\rho}_{N-1}(k_{N-1})$, the last 2κ bits. of $\rho(k_{N-1})$. If $B = B'$, then $\tilde{\rho}_{N-1}(k_{N-1}) = \tilde{\rho}_{N-1}(k'_{N-1})$. Since the adversary has done less than 2^κ work, he has only a negligible chance of finding $k_{N-1} \neq k'_{N-1}$ with the last 2κ bits of $\rho(k_{N-1})$ and $\rho(k'_{N-1})$ equal. Thus, except with negligible probability, $k_{N-1} = k'_{N-1}$.

Now consider the block of 2κ bits before the final block of 2κ bits of B. This is just $\tilde{\rho}_{N-2}(k_{N-1}) \oplus \tilde{\rho}_{N-1}(k_{N-2})$. So if $B = B'$, and $k_{N-1} = k'_{N-1}$ as above, then we must have that $\tilde{\rho}_{N-1}(k_{N-2}) = \tilde{\rho}_{N-1}(k'_{N-2})$, and as above, $k_{N-2} = k'_{N-2}$. Continuing in this way, we get that $(k_1, k_2, \ldots, k_{N-1}, k_\mu) = (k'_1, k'_2, \ldots, k'_{N-1}, k'_\mu)$ except with negligible probability.

Note that this logic would not have extended to k_0, had it been included, since only κ, and not 2κ, bits of $\rho(h_\rho(\alpha_0^{x_0}))$ are included in β_N.

Finally, if $B = B'$ and $(k_1, k_2, \ldots, k_{N-1}, k_\mu) = (k'_1, k'_2, \ldots, k'_{N-1}, k'_\mu)$, then clearly $p_0 = p'_0$.

Let $\mathbf{k} = (k_1, \ldots, k_{N-1}, k_\mu)$. Then we just showed that the function $g(p_0, \mathbf{k})$ is indistinguishable from a true random oracle with less than 2^κ work. Now let $f_{p_0}(\mathbf{k}) = g(p_0, \mathbf{k}) \oplus \hat{\rho}_1(k_1) \oplus \cdots \oplus \hat{\rho}_{N-1}(k_{N-1})$. Since the $\hat{\rho}_i$ do not call the random oracle μ, this is also indistinguishable from a true random oracle.

But if the adversary constructed a mix message which was successfully processed by $n_0, \ldots, n_N$, then he has a solution to $\hat{\rho}_0(k_0) = f_{\rho'_0(k_0)}(\mathbf{k})$, where $\rho'_0(x) = \rho(x)_{[(2r+2)\kappa..(2r+3)\kappa-1]}$; namely, $k_i = h_\rho(\alpha_i^{x_i})$ and $k_\mu = h_\mu(\alpha_N^{x_N})$. But this is just problem P.

Therefore, since he has only a negligible probability of finding a solution to problem P with considerably less than 2^κ work, he also has only a negligible chance of constructing a mix message which will be successfully processed by $N+1$ nodes, and the result is proven.

It is instructive to note where this proof relies on the fact that $N > r$. The key is that for $i \leq r$, the computation of γ_i contains bits from β_0. For example, $\gamma_r = \left(\bigoplus_{j=0}^{r-1} \hat{\rho}_{j+1}(h_\rho(\alpha_j^{x_j}))\right) \oplus \beta_{0[(2r+1)\kappa..(2r+2)\kappa]}$. [Compare this to γ_N, above, which equalled $\bigoplus_{j=0}^{N-1} \hat{\rho}_j(h_\rho(\alpha_j^{x_j}))$, with no component from β_0.]

Since the computation of β_r does not involve these bits of β_0, it is easy to find $\alpha_0^{x_0}, \ldots, \alpha_{r-1}^{x_{r-1}}$ and β_0 that satisfy $\gamma_r = \mu(h_\mu(\alpha_r^{x_r}), \beta_r)$. Just pick any $\alpha_0^{x_0}, \ldots, \alpha_{r-1}^{x_{r-1}}$, compute β_r, and let $\beta_{0[(2r+1)\kappa..(2r+2)\kappa]} = \mu(h_\mu(\alpha_r^{x_r}), \beta_r) \oplus \left(\bigoplus_{j=0}^{r-1} \hat{\rho}_{j+1}(h_\rho(\alpha_j^{x_j}))\right)$.

4.3. Wrap-resistence

We need to show that given a mix message $((\alpha', \beta', \gamma'), \delta')$, an adversary is unable to *wrap* it; that is, the adversary cannot produce a mix message $((\alpha, \beta, \gamma), \delta)$ such that a mix node (*even one whose private key x the adversary can select*) processing $((\alpha, \beta, \gamma), \delta)$ will yield $((\alpha', \beta', \gamma'), \delta')$.

In order for the adversary to succeed, it is necessary that $\alpha^{h_b(\alpha,s)} = \alpha'$ where $s = \alpha^x$. We will show that an adversary which makes c queries to the random oracle h_b can find such an (α, x) pair with probability at most $\frac{c}{q-1}$; for an adversary that does less than 2^κ work, this is negligible.

The proof is simple: if the adversary outputs a correct (α, x) pair, then she must have queried the random oracle with (α, α^x). But each (α, s) query to the oracle yields a random value $b \in_R \mathbb{Z}_q^*$. Since α is a generator of $\mathcal{G}^*$, the probability that α^b equals the given α' is $\frac{1}{q-1}$, and the result follows.

4.4. Security and Indistinguishability of Forward and Reply Messages

We need to show that an adversary controlling all nodes except one particular node, N, cannot distinguish mix messages entering node N, where each contains a (Δ, m) pair of the adversary's choice, each has a path following node N of the adversary's choice, and the messages can each be either forward or reply messages, as the adversary likes. Additionally, the adversary can see how N reacts to any mix message except one with a header matching the challenge message.

In particular, this would not only prove that the security property of [5] is satisfied, but also that forward and reply messages are indistinguishable to

any party that does not know the exit node's private key.

Formally: consider the following game $\mathbf{G}$. The adversary selects a sequence of mix nodes $n_0, \ldots, n_{\nu-1}$ with $\nu \leq r$. One of these, n_j, is the challenge node N. The adversary can select the private keys $x_{n_i} \in \mathbb{Z}_q^*$ for all $i \neq j$, but does not know x_{n_j} (he does know the public key y_{n_j}). The adversary also selects a destination address $\Delta \in \mathcal{D}\backslash\{*\}$, a message m, and a bit f that indicates whether this message should be a forward message ($f = 1$) or a reply message ($f = 0$).

The adversary then selects a second set $n'_0, \ldots, n'_{\nu'-1}$ with $\nu' \leq r$, and $\Delta' \in \mathcal{D}\backslash\{*\}$, m', and f'. It must be the case that $(n_0, n_1, \ldots, n_j) = (n'_0, n'_1, \ldots, n'_j)$, but it need not be the case that the list of subsequent nodes after $n_j = n'_j = \text{N}$ is the same (or even of the same length).

The challenger randomly chooses a bit b and constructs one of two mix messages as follows:

If b=0 and f=0: The challenger passes Δ and $\{n_0, \ldots, n_{\nu-1}\}$ to the procedure of section 3.4 to create a single-use reply block. There is no need to store values in the local table, or to send $(n_0, M_0, \tilde{k})$ to the nymserver. However, M_0 and $\tilde{k}$ are used, along with m, in the procedure of section 3.5 to construct the mix message $((\alpha_0, \beta_0, \gamma_0), \delta_0)$ which it would send to n_0.[1]

If b=0 and f=1: The challenger passes Δ, m, and $\{n_0, \ldots, n_{\nu-1}\}$ to the procedure of section 3.3. This procedure returns the mix message $((\alpha_0, \beta_0, \gamma_0), \delta_0)$.

If b=1: The challenger performs the same actions as above, but uses the primed values $n'_0, \ldots, n'_{\nu'-1}$, Δ', m' and f' instead of their unprimed counterparts.

$((\alpha_0, \beta_0, \gamma_0), \delta_0)$ is given to the adversary, whose job it is to determine b. The adversary can also give any mix message $((\alpha', \beta', \gamma'), \delta')$ to the challenge node n_j to see how it reacts, so long as $(\alpha', \beta', \gamma') \neq (\alpha_j, \beta_j, \gamma_j)$. Here, as in [5, §4.2], we only care about uniqueness of the header, not of the message body.

We will show that the adversary cannot determine the value of b with significantly better chance than random guessing. Once we have proven this, we note that the ability of the adversary to individually select whether each of the two messages is a forward message or a reply message also implies our desired property that the adversary cannot distinguish forward messages from replies. This holds so long as there is even a single node yet to process the message whose private key the adversary does not know.

The *advantage* of the adversary is the difference between $1/2$ and the probability the adversary guesses b correctly. We wish to show that the advantage for an adversary that does significantly less than 2^κ work is negligible.

We use the usual method of hybrid games. We first note that since the adversary can select the private keys $x_{n_0}, \ldots, x_{n_{j-1}}$, without loss of generality, we can assume that $j = 0$.

Game $\mathbf{G}_1$ is the same as $\mathbf{G}$ except that s_0 (in the procedure of section 3.2, called from section 3.3 in the case of a forward message or section 3.4 in the case of a reply message) is selected uniformly at random from $\mathcal{G}^*$, as opposed to being calulated as $s_0 = y_{n_0}^x$. An adversary that can distinguish game $\mathbf{G}$ from $\mathbf{G}_1$ can easily be used to distinguish $(y_{n_0}, \alpha_0 = g^x, s_0 = y_{n_0}^x)$ from (y_{n_0}, α_0, z) for a random $z \in_R \mathcal{G}^*$, thus solving the DDH problem in $\mathcal{G}^*$, contrary to our choice of $\mathcal{G}^*$. Here it is important that the adversary should not be allowed to query N with the challenge $(\alpha_0, \beta_0, \gamma_0)$ header, since N would not be able to process it. That γ must be a MAC on β with key $h_\mu(\alpha_0^{x_{n_0}})$ ensures that $(\alpha_0, \beta, \gamma) \neq (\alpha_0, \beta_0, \gamma_0)$ will be rejected by N except with negligible probability. If any (α, β, γ) is submitted to N with $\alpha \neq \alpha_0$ such that N successfully processes the message, then the success of the MAC ensures that, again except with negligible probability, the adversary knew the MAC key $h_\mu(\alpha_0^{x_{n_0}})$. Since h_μ is a random oracle, the adversary must have queried it with $\alpha_0^{x_{n_0}}$. But if the adversary knows that last value, he can process the message just as well as N can, and the ability to query N does not help him.

Game $\mathbf{G}_2$ is the same as $\mathbf{G}_1$ except that β_0, γ_0, and δ_0 are selected uniformly at random from their respective domains. If the adversary can distinguish games $\mathbf{G}_1$ and $\mathbf{G}_2$, then he can distinguish (with less work than 2^κ) the output of ρ with a random input from a random string, or μ with a random key from a random function, or π with a random key ($h_\pi(s_0)$—with s_0 being the randomly selected value from game $\mathbf{G}_1$—in the case of forward messages, or $\tilde{k}$ in the case of reply

1. Following section 3.5, we assume for simplicity that all reply messages are delivered using the nymserver. However, this assumption is not essential. If Alice wishes to send a reply block directly to Bob, for Bob's use in replying to her, she just modifies the procedure of section 3.4 to send him (n_0, M_0), omits $\tilde{k}$ from the tuple in her local table, and omits the $\pi^{-1}(\tilde{k}, \cdot)$ step from the procedure of section 3.7. The proof then need only have one additional part: to show that an adversary cannot cryptographically distinguish replies output by the nymserver from replies output by a first node n_0 whose private key the adversary does not know. (Here, "cryptographically distinguish" excludes distinuguishing based on traffic analysis; that is, observing the origin of the message.) The remaining path in the two reply blocks should be the same, and the adversary is allowed to know all other nodes' private keys. This is straightforward: the only salient difference between the messages is that the payload in the nymserver message is $\pi(\tilde{k}, 0_\kappa \| m)$ and in the other is $\pi^{-1}(h_\pi(s_{n_0}), 0_\kappa \| m)$. Both of these are indistinguishable from a random string to an adversary that knows neither $\tilde{k}$ nor s_{n_0}.

messages) from a random permutation, which he can do with only negligible probability, by our choice of ρ, μ, and π.

In game $\mathbf{G}_2$, since α_0 is independent of the bit b, and β_0, γ_0, and δ_0 are all random (and independent of b), it is clear the adversary's advantage is 0. Since each game is indistinguishable from the one before to the adversary, except with negligible probability, we see that the adversary's advantage in the original game $\mathbf{G}$ is negligible, as required.

5. Performance and Space Efficiency

In this section we give a brief overview of established cryptographic packet formats, and compare them to Sphinx both in terms of functionality as well as message size overhead. Throughout this section p denotes the size of any public key element in a packet format, s denotes the size of the symmetric key elements (per hop), and r denotes the maximum number of hops that messages can be routed through (all sizes are in bytes). When comparing overhead sizes, we will attempt to match the 128-bit security offered by Sphinx. Some older designs only supplied 80-bit security, using 1024-bit RSA keys, for example. We will be generous to the competing formats and stipulate that an RSA or Diffie-Hellman modulus of 2048 bits (256 bytes) is sufficient to offer 128-bit security, even though NIST [2] suggests that 3072-bit moduli are more appropriate for that security level. In the elliptic curve setting, we use the usual figure of 256-bit (32-byte) elements (assuming only the x-coordinate of the elliptic curve point is required, or point compression is used) in order to achieve 128-bit security.

The Sphinx packet format relies on a single public key element, blinded at each stage of mixing, and for each hop a message authentication code and the appropriate routing information. The cryptographic overhead sums to $p + (2r + 1)s$ bytes in total for the header, and an additional s bytes for integrity of the payload. The most costly operations involved in building a packet are the $2r$ public key operations. Relaying a Sphinx message requires only two public key operations (the Diffie-Hellman and blinding operations), plus the check that $\alpha \in \mathcal{G}^*$, for some choices of $\mathcal{G}^*$. (In Curve25519, for example, this check does not involve a public-key operation, but in $\mathbb{Z}_p^*$, it does.)

Mixmaster [16] is an established remailer infrastructure, with about 25 nodes with over 90% reliability according to Echolot statistics[2] as of November 2008. The main cryptographic shortcoming of the format is the lack of support for anonymous replies. The now-aging design uses 1024-bit RSA for the asymmetric encryption part, encapsulating routing information and a 3-DES key to be used in CBC mode (with a changing IV). Integrity is ensured through the use of an MD5 hash. The standard supports relaying messages over 20 hops—each hop adding a fixed 512 bytes of overhead. In total the equivalent of the routing header occupies 10240 bytes. Abstracting away from the concrete cryptographic mechanisms employed the length of a mixmaster header is $(1 + p + 8 + 8s + 31)r + s$ bytes (including a version number, an IV and padding). The Sphinx header is shorter since it does not require a public key element for each hop, and does not require an IV, since each key is used only once.

The first provable cryptographic packet format was proposed by Möller [15] and provided very similar functionality to the Mixmaster format—it supports sender-anonymous messages but not replies. The scheme makes use of multiple layers of DHAES/DSIES encryption. This requires an element of a cyclic group on which the Decisional Diffie-Hellman problem is hard, and a message authentication code *per hop* of mixing. To achieve a similar level of security to Sphinx this introduces an overhead of 272 bytes (256 bytes for the group element and 16 bytes for the MAC) per hop in addition to any routing information. The length of a packet header can be abstracted as $(p + s)r$. As for Mixmaster the format length suffers from the fact that a separate public key element has to be included for each stage of mixing.

Camenisch and Lysyanskaya [5] provided formal definitions in the Universal Composability model, and a concrete, provably secure, packet format for mixing forward messages, which we call the CL05 format. The CL05 scheme separates the messages into a header and a payload. The header is composed of asymmetrically encrypted ciphertexts containing the address of the next hop. They rely on a CCA2-secure encryption scheme with tags, and the bulk of the encryption is performed using a pseudorandom permutation (that is, a block cipher). The length of the header in the CL05 scheme is $rp + (r + 1)s$—unsurprisingly very similar to Möller [15], given the shared design philosophy. It is worth noting that the CL05 scheme does support replies, with a very similar cost, but those are *distinguishable* from sender anonymous messages.

Mixminion [8] was the first packet format to propose indistinguishable anonymous replies. Its cryptographic design is based on two headers which are swapped midway along the path. The headers are constructed using layers of asymmetric encryption (2048-bit RSA-

2. http://www.palfrader.org/echolot/

Scheme	Overhead Length	Indistin. Replies	Security	$\mathbb{Z}_n^*$ (p=256, s=16, r=5)	ECC (p=32, s=16, r=5)
Mixmaster [16]	$(1+p+8+8s+31)r+s$	no	heuristic	2136	1176[1]
Möller [15]	$(p+s)r$	no	provable	1360	400[1]
Mixminion [8]	$2[p+(2s+2s)(r-1)]+s$	yes	heuristic	1040	848[1]
CL05 [5]	$rp+(r+1)s$	no	provable	1376	416[1]
Minx [9]	$p+(s+2)(r-1)$	yes	broken	328	232[1]
SSH08 [21]	pr	yes	provable	1280	160[2]
Sphinx	$\mathbf{p+(2r+2)s}$	**yes**	**provable**	**448**	**224**

Table 1. Comparison between the lengths of different cryptographic packet formats in bytes. RSA schemes superscripted with ([1]) have been converted to use elliptic curve Elgamal (hence $p' = 2p$), while schemes superscripted with ([2]) were modified to use a simple Diffie-Hellman over EC. The parameter p denotes the length of asymmetric elements, s is the length of symmetric elements, while r is the maximum path length.

OAEP) and AES in CBC mode. Instead of appending the RSA ciphertexts to each other, Mixminion compresses the headers by including parts of the next RSA ciphertext in the plaintext of the previous one. This means that the 2048 bytes of each sub-header can encode information for more than 8 hops. Given that the OAEP overhead is about $2s$ and that a hash and a key is contained in each layer of the header (again $2s$) the cost of both headers of Mixminion is $2[p+(2s+2s)(r-1)]+s$ bytes. There exist no known attacks against this format, but at the same time it only comes with heuristic security arguments.

Minx [9] was the first attempt to achieve a very compact mix packet format. It uses raw RSA, and AES in IGE and bi-IGE modes, without any additional overhead for integrity checking. Instead it relies on the fragility and error propagation characteristics of bi-IGE to ensure no information is recoverable from tagging attacks. Like Mixminion, it encapsulates parts of RSA ciphertexts into previous plaintexts, making the headers quite small. Its abstract length is $p+(s+1)(r-1)$ bytes (assuming only a single byte of routing data). The security argument underlying Minx is heuristic, however, and recent work [21] shows that there is indeed a polynomial-time attack against it, taking advantage of the naive use of raw RSA encryption.

Shimshock *et al.* [21] proposed a fix for Minx, which we denote SSH08. The SSH08 format encodes only keys in separate RSA headers destined to each mix, and for technical reasons does not use the compression technique employed by Mixminion and Minx. This leads to a cost of pr bytes. It is also worth noting that the encoding of a message relies on making one byte of the hash of the RSA plaintext collide with the one byte destination of the packet. That is, there is no way of telling hop i which node should be hop $i+1$ short of constructing a message whose hash happens to contain the (8-bit) identifier of the desired hop $i+1$. This requires, on average, about 256 RSA encryptions *per hop* to construct the message. This design choice makes the packet format quite compact, and easily portable to elliptic curves, at the cost of flexibility. The sender can only communicate a very small amount of information to each mix, since it would have to find—using brute force—an element that decodes to the desired information.

Table 1 summarises the overhead lengths and other properties of each cryptographic packet format and compares them to Sphinx. Concrete lengths are illustrated through the choice of two sets of parameters, in each case for a message that is capable of travelling on paths up to length $r = 5$. First the length of the header is calculated for cryptosystems based on the hardness of the discrete logarithm and RSA problems over number fields. To achieve 128-bit security we require $p = 256$ byte (2048-bit) asymmetric elements, and $s = 16$ byte symmetric elements. We can see that in this context Sphinx outperforms all other secure proposals (note that Minx is not considered secure [21]).

A second comparison is made between Sphinx and other schemes when they are implemented using asymmetric primitives based on elliptic curves. In that context we substitute RSA encryption, used by most other schemes, with an EC-based version of Elgamal. This requires two asymmetric elements of total length $p' = 2p$. Sphinx and SSH08 on the other hand do not encrypt anything, and the asymmetric part of the header is only required to perform key derivation. As such they only use a single element of length $p = 32$ bytes, the size necessary to achieve 128-bit security. The SSH08 scheme is more compact than Minx only because it is capable of transmitting very little information from the sender to the intermediate mixes (about 8 bits of information, at the cost of

about 2^8 public key operations per mix). Sphinx can carry much more information to intermediate mixes, supporting more complex mixing strategies, and at a much lower computational cost to the sender.

Finally, we note that Sphinx messages are computationally cheap to process: they require a single public key operation (an exponentiation in $\mathbb{Z}_n^*$ or multiplication in ECC) to derive a key, followed by fast symmetric-key operations.

6. Conclusions

Sphinx is a compact and provably secure mix packet format, designed as a drop-in replacement for existing remailers.

Sphinx is flexible in two important ways: first, it allows the system designer to chose their preferred family of cryptographic primitives. With Diffie-Hellman over prime fields, Sphinx is the most compact format, with an overhead of 448 bytes to route through 5 mixes. The use of ECC makes it even more compact, with a header length of only 224 bytes. The compactness of the format allows novel applications of anonymity to flourish: short messages can be cheaply routed, supporting privacy for services like microblogging with a low cryptographic overhead.

Second, Sphinx can be trivially extended to act as a general-purpose secure transport between senders of messages and the intermediate mixes on a path. This means that system designers are free to choose any mix strategies, including those that rely on the sender providing detailed information to mixes about the processing of messages. Such strategies are crucial for blending high-and-low latency traffic [11], or preventing flooding attacks [13].

The security guarantees associated with Sphinx are very strong, and backed by reduction proofs to well-studied cryptographic primitives. Hence, we can say with high confidence that the short length of the packets does not lead to any reduction in security.

The line of research up to this state-of-the-art Sphinx design demonstrates that the cryptographic aspects of anonymous communications are now well understood and mature. Off-the-shelf packet formats are now available to route messages through any system, allowing designers of anonymity systems to concentrate on preventing traffic analysis and sorting out denial-of-service problems, as well as implementing robust business models around anonymous communications. Sphinx can act as a transport layer for any of these applications, ensuring that the cryptography of the anonymous transport is no longer a security worry.

Acknowledgements

The design of Sphinx has greatly benefitted from discussions with Anna Lysyanskaya, at the 2005 Daghstul school on anonymous communications. Steven Murdoch, Greg Zaverucha and Emilia Käsper provided very useful early feedback on the ideas as well as a draft of the paper. We would like to thank the Natural Sciences and Engineering Research Council of Canada and the Mathematics of Information Technology and Complex Systems Network of Centres of Excellence for supporting this work. We would also like to thank the anonymous reviewers for making suggestions to improve this paper.

References

[1] Ross J. Anderson and Eli Biham. Two Practical and Provably Secure Block Ciphers: BEARS and LION. In Dieter Gollmann, editor, *FSE*, volume 1039 of *Lecture Notes in Computer Science*, pages 113–120. Springer, 1996.

[2] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for Key Management—Part 1: General (Revised). National Institute of Standards and Technology Special Publication 800-57, May 2006.

[3] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In *Public Key Cryptography 2006*, pages 207–228, 2006.

[4] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 30–45. Springer-Verlag, LNCS 2009, July 2000.

[5] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *Proceedings of CRYPTO 2005*, pages 169–187. Springer-Verlag, LNCS 3621, August 2005.

[6] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

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

[8] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, May 2003.

[9] George Danezis and Ben Laurie. Minx: A simple and efficient anonymous packet format. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)*, Washington, DC, USA, October 2004.

[10] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[11] Roger Dingledine, Andrei Serjantov, and Paul Syverson. Blending different latency traffic with alpha-mixing. In George Danezis and Philippe Golle, editors, *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, pages 245–257, Cambridge, UK, June 2006. Springer.

[12] Ceki Gülcü and Gene Tsudik. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96*, pages 2–16. IEEE, 1996.

[13] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of Information Hiding Workshop (IH 1998)*. Springer-Verlag, LNCS 1525, 1998.

[14] David Mazières and M. Frans Kaashoek. The Design, Implementation and Operation of an Email Pseudonym Server. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS 1998)*. ACM Press, 1998.

[15] Bodo Möller. Provably secure public-key encryption for length-preserving chaumian mixes. In *Proceedings of CT-RSA 2003*. Springer-Verlag, LNCS 2612, 2003.

[16] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. IETF Internet Draft, 2003.

[17] Mridul Nandi and Douglas R. Stinson. Multicollision Attacks on Some Generalized Sequential Hash Functions. *IEEE Transactions on Information Theory*, 53(2):759–767, February 2007.

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

[19] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[20] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In Fabien Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.

[21] Eric Shimshock, Matt Staats, and Nick Hopper. Breaking and Provably Fixing Minx. In Nikita Borisov and Ian Goldberg, editors, *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 99–114, Leuven, Belgium, July 2008. Springer.