Using Sphinx to Improve Onion Routing Circuit Construction*

Aniket Kate and Ian Goldberg

David R. Cheriton School of Computer Science
University of Waterloo, ON, Canada
{akate,iang}@cs.uwaterloo.ca

Abstract

This paper presents compact message formats for onion routing circuit construction using the Sphinx methodology developed for mixes. We significantly compress the circuit construction messages for three onion routing protocols that have emerged as enhancements to the Tor anonymizing network; namely, Tor with predistributed Diffie-Hellman values, pairing-based onion routing, and certificateless onion routing. Our new circuit constructions are also secure in the universal composability framework, a property that was missing from the original constructions. Further, we compare the performance of our schemes with their older counterparts as well as with each other.

1 Introduction

Chaum [10] introduced the concept of *digital pseudonyms* and *mix networks* in 1981. Extending this seminal work, Goldschlag, Reed and Syverson [21] proposed *onion routing* to achieve low-latency anonymous communication on public networks, which motivated the original Onion Routing project [35, 22, 40] and many other anonymous communication constructions [11, 18, 36, 16]. Among these, with its hundreds of thousands of users, the second generation onion routing project—Tor [41]—has turned out to be a huge success. However, with its latency times of a few seconds, users find Tor to be very slow for their usual communication over the Internet, and employ it only in situations where their anonymity is indispensable to them. Efficiency is essential for widespread use of anonymity networks; therefore, defining an efficient practical onion routing protocol forms the motivation of this work.

An onion routing (OR) network consists of a set of *onion routers* (OR nodes) that relay traffic, a large set of users and a *directory server* that provides routing information of the OR nodes to the users. A user constructs a *circuit* choosing a small ordered subset of OR nodes, where the chosen nodes route the user's traffic over the path formed. The key property is that it is difficult for any OR node in a circuit to determine the circuit nodes other than its predecessor and successor. Further, the task must also be difficult for a powerful but not global observer. The user achieves this by sending the first OR node an *onion*—a message wrapped in multiple layers of encryption (one layer per selected node). Although OR circuit construction is a simple concept, the task of simultaneously achieving efficiency and security presents an interesting research challenge.

In the original Onion Routing circuit construction [35], a user includes the identifier of the next node and a random symmetric session key in each onion layer, and uses nodes' public keys to

*This is the full version of our paper appearing in the 14th International Conference on Financial Cryptography and Data Security (FC 2010) [25].

encrypt their respective layers. A node decrypts a received onion using its private key, forwards the remaining onion to the next node, and uses the random symmetric session key for the rest of the session. However, this *single-pass* circuit construction is not *forward secret*; if an adversary corrupts a node and obtains its private key, then the adversary can decrypt all of its past communication. The adversary could then successively compromise all the nodes in a circuit to break the anonymity of a user's past communications. Although changing the the public/private key pairs for all OR nodes after a predefined interval (*forward secrecy phase*) is a possible solution, it is not scalable. Every system user now has to download a new set of public keys for all the nodes at the start of every forward secrecy phase.

Observing the above issue with forward secrecy, Dingledine, Mathewson and Syverson [16] used an interactive and incremental *telescoping* approach while designing Tor. In the Tor authentication protocol (TAP), which is used to negotiate the session keys in this *multi-pass* circuit construction, a node's public key is only used to initiate the construction and its compromise does not void the security of the session keys once the randomness used in the protocol is erased. Øverlier and Syverson [33] improved the efficiency of Tor using a half-certified Diffie-Hellman (DH) key agreement [29, §12.6] in their Tor with predistributed DH values (Tor-preDH).

However, in Tor, $\Theta(\nu^2)$ messages are required to create a circuit of length ν, as compared to $\Theta(\nu)$ required in a single-pass circuit construction. To solve the scalability issue in single-pass circuit constructions, Kate, Zaverucha and Goldberg [26, 27] suggested the use of an identity-based setting and defined a pairing-based onion routing protocol (PB-OR). Catalano, Fiore and Gennaro [9] suggested the use of a certificateless setting instead and defined two certificateless onion routing protocols (CL-OR and 2-CL-OR). Øverlier and Syverson [33] have also suggested a single-pass circuit construction that provides forward secrecy eventually. However, an extensive comparison between all these schemes is not available yet. In terms of security, [20] proved the security of TAP, while [27] and [9] proved some security properties of PB-OR and CL-OR respectively. However, none of these practical protocols achieve security in the universal composability (UC) framework [8]. Camenisch and Lysyanskaya [7] presented a framework for UC-secure OR circuit construction, but their protocol is not practical enough for realistic use.

1.1 Contributions

In this paper, we present a practical generic onion routing circuit construction protocol that achieves security in the UC model. We apply our protocol to Tor-preDH, PB-OR, CL-OR and 2-CL-OR to define their UC-secure versions. Importantly, the circuit construction messages for these new protocols are significantly smaller than those in the original protocol and there is no addition to a user's computational cost. We achieve this using *Sphinx*, an efficient message format for mix networks, defined by Danezis and Goldberg [13].

We first review the Sphinx message format and describe various OR circuit construction protocols (§2). We then give a generic Sphinx-based OR circuit construction protocol and discuss its security properties (§3). Next, we use this generic message format to define practical UC-secure circuit constructions (§4). We compare the message compactness and computational cost of the new constructions with their original counterparts (§5). Finally, we compare these three onion routing protocols in terms of computational cost and systems issues (§6).

2 Preliminaries

2.1 Notation

In this paper, we use Sphinx's notation and briefly describe it here. (Refer to [13, Sec. 3.1] for a detailed description.)

Our adversary is computationally bounded by a security parameter κ, and it has to do 2^κ computation to break the security of any of the protocols. Following the current security standards, we choose $\kappa = 128$. For a prime q of size 2κ bits, let $\mathbb{G}$ be a cyclic group of order q. In the original Sphinx algorithm, $\mathbb{G}$ is required to satisfy the *decisional Diffie-Hellman* (DDH) assumption [5]. In this paper, the required cryptographic assumption changes with the OR protocol under consideration; we defer their expositions to §4. Let ν be the length of the circuit. Sphinx makes the circuit construction message size independent of ν; we parametrize the maximum length of the circuit as r. Node identifiers are κ-bit strings. Each node has a public/private key pair. However, as the schemes that we are considering belong to three different methodologies (public-key cryptography—PKC, identity-based cryptography—IBC and certificateless cryptography—CLC), the formation of the public and the private keys changes for each scheme.

We assume a message authentication code (MAC) $\mu : \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \{0,1\}^\kappa$, a pseudo-random generator (PRG) $\rho : \{0,1\}^\kappa \rightarrow \{0,1\}^{(2r+3)\kappa}$ and corresponding hash functions $h_\mu, h_\rho : \mathbb{G}^* \rightarrow \{0,1\}^\kappa$, where $\mathbb{G}^*$ is the set of non-identity elements of $\mathbb{G}$. We use another hash function $h_b : \mathbb{G}^* \times \mathbb{G}^* \rightarrow \mathbb{Z}_q^*$ in the blinding factor computation. All hash functions are modelled as random oracles.

2.2 The Sphinx Message Format

Mix message formats have been a point of interest in research on mix networks [31, 30, 12, 14, 7, 38, 13] . Recently, Danezis and Goldberg [13] proposed Sphinx as the most compact and efficient cryptographic mix message format and proved its security in the UC model.

Cryptographically, the most elegant feature of the Sphinx message format is its session key derivation technique based on a repeatedly modified random element of a cyclic prime order group. We call this technique Sphinx's *blinding logic*. The mentioned random element (α) and its repeated modified forms are called *pseudonyms* since each of these random elements is a temporary public key whose private key is held by the user. In the Sphinx blinding logic, each mix node uses a pseudonym supplied by its predecessor and its own private key to compute the session key with the user. To improve the unlinkability, as in Tor, a pseudonym must not remain the same across the circuit. In the mix network and onion routing literature, this is done by including separate random pseudonyms in a construction message for each node in the circuit. In Sphinx's blinding logic, this is achieved using a single repeatedly changing pseudonym. At every node, a blinding factor is extracted from the current pseudonym and the newly computed session key. The current pseudonym is then exponentiated with the blinding factor to generate the next pseudonym. In other words,

$$\alpha_{i+1} = \alpha_i^{h_b(\alpha_i, s_i)}. \tag{1}$$

The session key is computed by node n_i as $s_i = \alpha_i^{x_i}$, where x_i is the node's private key, and by the user as explained in §4.

To send an anonymous message, a sender first chooses her mix nodes and obtains their public keys. She then computes α_i and s_i and wraps the message in multiple layers of encryption using the PRG ρ to generate ciphertext values β_i. To check the integrity of the message header, she calculates and includes a MAC γ_i at each mixing stage. Upon receiving a message header $(\alpha_i, \beta_i, \gamma_i)$, each mix node n_i extracts session keys using its private key x_i and the pseudonym α_i received from the predecessor. It uses those to verify the MAC γ_i and to decrypt a layer of encryption of β_i. It also extracts the routing information, computes the pseudonym α_{i+1} for the next node (Equation 1) and forwards the message to n_{i+1}. As we are working towards using Sphinx in OR circuit constructions, we ignore Sphinx's payloads and reply blocks.

Note that we are here concerned only with the circuit construction messages. Our methodologies are not required for OR communications in already constructed circuits.

2.3 Tor Circuit Construction

In Tor circuit construction [16], a user performs a DH key agreement [15] with each successive node in her circuit over a secure tunnel formed using the already-agreed session keys. This ensures the forward secrecy of the communication immediately after these session keys are deleted. In the Tor authentication protocol (TAP), a user extends a circuit to node n_i by generating a random $x_i \in_R \mathbb{Z}_q^*$ and sending a DH value (that is, a pseudonym) g^{x_i} encrypted using the (RSA) public key of node n_i. Node n_i decrypts the message and responds by sending g^{y_i}, where $y_i \in_R \mathbb{Z}_q^*$, and a hash of $g^{x_i y_i}$. It is important that the user herself generates and encrypts the DH value g^{x_i}; if an intermediate adversary OR node (n_j for $0 < j < i$) derives g^{x_i}, it can launch a *man-in-the-middle* attack. If g^{x_i} is available to node n_{i-1} in the clear, it can generate $x_i' \in_R \mathbb{Z}_q$ and send $g^{x_i'}$ to n_i. On the other hand, it can generate $y_i' \in_R \mathbb{Z}_q$, $g^{y_i'}$ and $g^{x_i y_i'}$ and send $g^{y_i'}$ as well as a hash of $g^{x_i y_i'}$ to the user. Node n_{i-1} can now unwrap any message the user sends to node n_i. It can then re-wrap it appropriately and send it to n_i, making the attack completely undetectable.

In Sphinx's blinding logic, node n_{i-1} uses the received pseudonym $g^{x_{i-1}}$ to generate and send pseudonym g^{x_i} to node n_i unencrypted. Therefore, it is not possible to directly apply the compact Sphinx message format to Tor.

2.4 Recent Enhancements Suggested for Tor

Although widely deployed, Tor is still not a part of everyday web access for most of its users because of its slowness. Along with the network issues such as TLS congestion control [34], the telescoping circuit construction is also considered to be an important reason behind this inefficiency. [33, 27]

Øverlier and Syverson [33], Kate *et al.* [26], and more recently Catalano *et al.* [9] suggested improvements to Tor circuit construction. These schemes use a *one-way anonymous* key agreement [26] strategy in the public-key cryptography (PKC), identity-based cryptography (IBC) and certificateless cryptography (CLC) settings respectively. Here, a user chooses a random element of $\mathbb{Z}_q^*$ per circuit node and computes an associated pseudonym. A session key is computed using the node's public key and the random element at the user end, and using the pseudonym received and the node's private key at the node's end; the precise session-key computation and the cryptographic assumption vary with the OR circuit construction protocol. Most importantly, unlike Tor, the user does not encrypt the pseudonyms in these schemes, which is a direct result of the inclusion of the private key of an OR node in the session key generation. Therefore, it is possible to incorporate Sphinx's blinding logic into these schemes.

3 Generalizing the Sphinx Methodology

In this section, first we discuss our design goals and threat model. We then present the generic design of OR circuit construction using the Sphinx methodology and discuss its security properties in the UC model.

3.1 Design Goals and Threat Model

Our design goals and threat model are same as that of Tor. It must be difficult for the adversaries to link multiple communications to or from a single user. It also should not be feasible for any node to determine the identity of any node in a circuit other than its two adjacent non-adversary nodes. Although it is impossible to prevent denial-of-service attacks by an adversary node, the circuit construction protocol should be able to detect any malicious modification to circuit construction messages.

We do not consider a global passive adversary, which in theory can follow end-to-end traffic. Our adversaries have complete control over some part (but not all) of the network, as well as control over some of the nodes. Furthermore, we assume that the adversaries are mobile and may shift to the currently unadversarial nodes in the future. Thus, the circuit construction protocols should be forward secret so that after some time, the session keys used to protect node identities and the contents of messages are irrecoverable even if all participants in a circuit are subsequently compromised.

3.2 Generic Design

In Sphinx, a pseudonym α_{i+1} for node n_{i+1} is generated using the pseudonym α_i and the session key s_i generated at node n_i (Equation 1). As discussed in §2.3, we can use the Sphinx methodology in an OR circuit construction protocol where a node can create or observe a pseudonym for the next node in the circuit. We here present a generic OR circuit construction using the Sphinx methodology.

For the OR circuit construction, we slightly modify the Sphinx notation from §2.2. We modify the definition of PRG ρ to be $\rho : \{0,1\}^\kappa \to \{0,1\}^{(2r+1)\kappa}$. In the original Sphinx format, a destination address (Δ) and a reply block identifier $I \in \{0,1\}^\kappa$ are present. Since we do not require circuit construction messages to be delivered to external parties, we can remove these portions of the Sphinx format, saving 2κ bits in the size of β_i. As the sets to which the session keys s_i belong change with the OR protocol used, we also modify the definition of our hash functions as follows: $h_\mu, h_\rho : \mathbb{G}' \to \{0,1\}^\kappa$ and $h_b : \mathbb{G}^* \times \mathbb{G}' \to \mathbb{Z}_q^*$, where $\mathbb{G}'$ represents a set of values that the session key s_i can take.

To create an OR circuit construction message, we use Sphinx's mix header creation algorithm ([13, §3.2]) with a generalization of the session key generation. The original Sphinx message format is based on the half-certified DH key agreement [29, §12.6], where a session key s_i is generated as $s_i = y_i^{xb_0b_1\cdots b_{i-1}}$ at the user's end and as $s_i = \alpha_i^{x_i}$ at node n_i, where (y_i, x_i) is the public/private key pair for n_i, α_i is a pseudonym for node n_i, x is the session-specific randomness and $b_0, \ldots, b_{i-1}$ are the blinding factors, $b_i = h_b(\alpha_i, s_i)$. The different OR circuit construction protocols use different session key generation methods, so we generalize this session key generation step. At the user end, we set

$$s_i = f_U(y_i, xb_0b_1\cdots b_{i-1}), \tag{2}$$

and at node n_i,

$$s_i = f_N(x_i, y_i, \alpha_i). \tag{3}$$

The other technical details of Sphinx remain exactly the same. Refer to [13, §3.2] for circuit construction message creation and to [13, §3.6] for message processing at a node.

Note that, although Sphinx is defined for single-pass constructions, its blinding logic is also useful in multi-pass constructions, where it can avoid the transfer of pseudonyms in circuit extension messages. However, we concentrate only on single-pass constructions in this paper as the applicability of Sphinx is more evident there.

3.3 Security Analysis

Camenisch and Lysyanskaya [7] design a framework for onion routing with provable security in the UC model. They define onion-correctness, onion-integrity and onion-security properties for an OR scheme and prove Theorem 1.

Theorem 1 (Theorem 1 [7]) *An onion routing scheme satisfying onion-correctness, integrity and security, when combined with secure point-to-point secure channels, yields a UC-secure OR scheme.*

Danezis and Goldberg [13] separate a wrap-resistance property from onion-security to simplify the onion-security definition and prove the resulting four security properties of the Sphinx message format using random oracles. We use their security discussion to define the security requirements for our generic OR circuit design.

Onion-correctness: According to [7], an OR circuit construction is *correct*, if a message reaches the intended recipient in a constructed circuit whenever an onion is formed correctly, processed by the right routers in the right order, and these routers follow the protocol. It is easy to observe that a Sphinx-based OR circuit construction works correctly.

Onion-integrity: An OR circuit construction has *integrity* if it is not possible for an adversary to build a circuit of more than N honest nodes, for some predefined bound N, except with negligible probability. As we only change the session key generation step in the original Sphinx message format, our proof of integrity remains exactly the same as that of Sphinx in [13, Sec. 4.2] except in our case $N = r$ instead of $r + 1$ as we reduce the size of β_i by 2κ bits.

Wrap-resistance: Camenisch and Lysyanskaya [7] also suggest a security property such that it should not be possible for an OR node to wrap an existing onion; that is, given a target onion (say) O_{i+1}, it must not be possible for an adversary to construct onion O_i such that a node n_i processing O_i would output O_{i+1}. This must remain true even if the adversary can select n_i's private key. In Sphinx, the wrap-resistance property is based on the difficulty of computing α_i given $\alpha_{i+1} = \alpha_i^{h_b(\alpha_i, s_i)}$. As this blinding logic remains unchanged in our generic OR design, our wrap-resistance discussion is same as that of [13, Sec. 4.3].

Onion-security: As defined by [7], the *onion-security* property requires that an attacker controlling all but one honest node (say n^*) in a circuit should not be able to distinguish OR circuit construction messages entering into the unattacked node n^*. Onion-security for our generic construction differs in three places from the original Sphinx format.

1. There are no reply messages and there is no need for indistinguishability of forward and reply messages.

2. There are no destination addresses Δ and no message payloads.

3. Generation of the session key varies with the circuit construction protocol.

We observe that the first two differences do not affect the security proof. The third difference is related to the infeasibility of the adversaries to distinguish between s_i generated using Equations 2 and 3 and a random $s_i \in_R \mathbb{G}'$. The exact assumption required to achieve this indistinguishability property changes with the choice of f_U and f_N, and the OR circuit construction setting. Therefore, we here only ask that the adversaries should not be able to distinguish between a random s_i and s_i generated during the circuit construction and leave the discussion of the exact cryptographic assumptions required to the next section.

4 Applying the Generic Sphinx Format to OR Circuit Constructions

In this section, we implement the generic Sphinx format into three onion routing circuit constructions. We also discuss the exact security assumptions required for their UC model security.

4.1 Tor with Predistributed DH Values (Tor-preDH)

The half-certified DH key agreement scheme [29, §12.6] is a one-pass protocol with unilateral key authentication of the receiver to the sender, assuming that the sender has an authentic copy of the receiver's public key. Øverlier and Syverson [33] define an enhancement to the Tor circuit construction using half-certified DH key agreement instead of TAP.

Let $x_i \in \mathbb{Z}_q^*$ be the private key for node n_i and let $y_i = g^{x_i}$ be its public key, where $g \in \mathbb{G}$ is a chosen generator. In the half-certified DH key agreement scheme, a user generates a random $r_i \in_R \mathbb{Z}_q^*$ and sends a pseudonym $\alpha_i = g^{r_i}$ to node n_i over the already formed circuit (tunnel), if any. The user generates the session key as $s_i = y_i^{r_i}$ and node n_i generates $s_i = \alpha_i^{x_i}$. Øverlier and Syverson used this to present a single-pass protocol (their second protocol).

Using the generic Sphinx design presented in §3.2, we can not only make their eventual forward secret protocol more efficient but also prove its security in the UC model. Here, except for the entry node, the user is not required to send α_i to node n_i in the circuit. Node n_{i-1} generates the pseudonym $\alpha_i = \alpha_{i-1}^{h_b(\alpha_{i-1}, s_{i-1})}$. All other computation remains the same as the half-certified DH key agreement and the message format remains the same as that of Sphinx. As $s_i \in \mathbb{G}^*$, $\mathbb{G}' = \mathbb{G}^*$ here.

Security Analysis: Onion-security is the only security property of Sphinx that depends upon the underlying two-party key agreement protocol. The other security properties are independent and have been proven in §3.3. As both the original Sphinx protocol and the Tor-preDH construction use the half-certified DH key agreement protocol, their onion-security analyses also remain the same.

In brief, for onion-security, it should not be feasible for an adversary to distinguish $s_i = g^{x_i r_i} \in \mathbb{G}$ from a random element in $\mathbb{G}$, given g, $\alpha_i = g^{r_i}$ and $y_i = g^{x_i}$ except with negligible probability. It is easily possible to model any DDH problem $(g, g^a, g^b, z : z \stackrel{?}{=} g^{ab})$ in group $\mathbb{G}$ into this indistinguishability problem. Therefore, assuming that the DDH assumption holds in $\mathbb{G}$, our enhancement to Tor-preDH is secure in the UC model.

4.2 Pairing-Based Onion Routing

Kate *et al.* [26] observe that the public-key management issue while achieving forward secrecy in single-pass onion routing circuit constructions can be solved using IBC. They develop an anonymous key agreement protocol modifying Sakai-Ohgishi-Kasahara key agreement [37] in the Boneh-Franklin identity-based encryption (BF-IBE) setup [6] and use that to define an OR circuit construction called *pairing-based onion routing* (PB-OR).

We choose three cyclic groups $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ (all of which we shall write multiplicatively) of prime order q and a *bilinear pairing* $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. We refer the readers to the appendix for a more elaborate discussion of pairings. In the BF-IBE setup, given $(e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T,\ g \in \mathbb{G},\ \hat{g} \in \hat{\mathbb{G}})$, a (possibly distributed) private-key generator (PKG) generates a master key $s \in \mathbb{Z}_q^*$ and an associated public key $y = g^s \in \mathbb{G}^*$, and derives private keys d_i for nodes using their well-known identities and s. A node with identity $\mathtt{ID}_i$ receives the private key $d_i = (h_{\mathtt{ID}}(\mathtt{ID}_i))^s \in \hat{\mathbb{G}}^*$, where $h_{\mathtt{ID}} : \{0,1\}^* \to \hat{\mathbb{G}}^*$ is a cryptographic hash function.[1]

In PB-OR, a user generates a random $r_i \in_R \mathbb{Z}_q^*$ and sends a pseudonym $\alpha_i = g^{r_i}$ to node n_i over the already-formed circuit (if any). The session key s_i is generated at the user end as $s_i = e(y, h_{\mathtt{ID}}(\mathtt{ID}_i))^{r_i}$ and at the node n_i as $s_i = e(\alpha_i, d_i)$. Using our generic design, α_i can be generated as $\alpha_i = \alpha_{i-1}^{h_b(\alpha_{i-1}, s_{i-1})}$, while the computation of s_i remains the same as that of the original PB-OR, except here $r_i = x b_0 b_1 \cdots b_{i-1}$ for an $x \in_R \mathbb{Z}_q^*$ chosen by the user. As $s_i \in \mathbb{G}_T$, $\mathbb{G}' = \mathbb{G}_T$ here.

Security Analysis: As already mentioned, we only need to consider onion-security. In PB-OR with Sphinx, except with negligible probability, it should not be feasible for an adversary to distinguish $s_i = e(g, h_{\mathtt{ID}}(\mathtt{ID}_i))^{s r_i} \in \mathbb{G}_T$ from a random element of $\mathbb{G}_T$, given g^s, $\alpha_i = g^{r_i}$ and $h_{\mathtt{ID}}(\mathtt{ID}_i) \in \hat{\mathbb{G}}$. For the bilinear pairing tuple $(e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$, given an instance of the *decisional bilinear Diffie-Hellman (DBDH)* problem [23] $(g, g^a, g^b, \hat{g}, \hat{g}^a, \hat{g}^c, z : z \stackrel{?}{=} e(g, \hat{g})^{abc})$, it can be modelled as the above indistinguishability game using the following mapping: $\alpha_i = g^a$, $g^s = g^b$ and $h_{\mathtt{ID}}(\mathtt{ID}_i) = \hat{g}^c$. Therefore, assuming that the DBDH assumption holds for a tuple $(e, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$, our enhancement to PB-OR is secure in the UC model.

4.3 Certificateless Onion Routing

Catalano *et al.* [9] recently introduced the concept of certificateless onion routing and presented two protocols (CL-OR and 2-CL-OR) for it. Their motivation is to avoid pairings and to eliminate the interactions between a PKG (or key generation centre—KGC) and nodes in PB-OR using CLC introduced by Al-Riyami and Paterson [2].

In certificateless onion routing, the KGC chooses a random generator $g \in_R \mathbb{G}$, two hash functions $h_{CL} : \{0,1\}^* \to \mathbb{Z}_q$ and $h_\pi : \mathbb{G} \times \mathbb{G} \to \{0,1\}^\kappa$, and a master key $s \in_R \mathbb{Z}_q$. It then computes $y = g^s$ and publishes $(\mathbb{G}, g, y, h_{CL}, h_\pi)$ as the public key. When a node n_i with identity $\mathtt{ID}_i$ asks for its partial private key, the KGC first generates a random $k_i \in_R \mathbb{Z}_q$, computes $\omega_i = g^{k_i}$ and $z_i = k_i + h_{CL}(\mathtt{ID}_i, \omega_i)s$ and returns $d_i = (\omega_i, z_i)$ to node n_i. Each node also generates a random $t_i \in_R \mathbb{Z}_q$ and computes $u_i = g^{t_i}$. The public key for a node n_i with identity $\mathtt{ID}_i$ is (ω_i, u_i) and its private key is (z_i, t_i).

In CL-OR, a user generates a random $r_i \in_R \mathbb{Z}_q^*$ and sends the corresponding pseudonym $\alpha_i = g^{r_i}$ to node n_i. The user generates the session key $s_i = (z_{i1}, z_{i2})$ such that $z_{i1} = (\omega_i y^{h_{CL}(\mathtt{ID}_i, \omega_i)})^{r_i}$ and $z_{i2} = u_i^{r_i}$ and upon receiving pseudonym α_i, node n_i generates $z_{i1} = \alpha_i^{z_i}$ and $z_{i2} = \alpha_i^{t_i}$.

While incorporating the generic Sphinx design, only the computation of the pseudonym α_i changes in the above certificateless key agreement protocol. As above, the pseudonym α_i is generated as $\alpha_i = \alpha_{i-1}^{h_b(\alpha_{i-1}, s_{i-1})}$. As $s_i \in \mathbb{G} \times \mathbb{G}$, $\mathbb{G}' = \mathbb{G} \times \mathbb{G}$ here.

[1] In the PB-OR paper, the authors assume the *symmetric* (type 1) pairings. However, we maintain a generic definition as pairings of types 2 and 3 [19] are more efficient.

Security Analysis: Catalano *et al.* [9] discuss onion-security for their protocols. They show that their key agreement is onion-secure and its security is based the *strong Diffie-Hellman (SDH)* assumption [1]. As the onion-security property requires the indistinguishability of a session key from a uniformly random key, it is actually a property of the key agreement and is independent of the message format. Therefore, CL-OR with Sphinx message format is onion-secure under the SDH assumption and we refer the reader to [9, Sec. 3.2] for a detailed proof.

To avoid the SDH assumption, Catalano *et al.* also present another, slower, protocol (2-CL-OR) with security based on the weaker *computational Diffie-Hellman (CDH)* assumption [15]. We observe that in terms of using Sphinx in this protocol, only the s_i generation and the corresponding definition of $\mathbb{G}'$ changes in 2-CL-OR. Node n_i chooses $t_{i1}, t_{i2} \in_R \mathbb{Z}_q$ instead of $t_i \in_R \mathbb{Z}_q$ as its private key and the pair $(u_{i1} = g^{t_{i1}}, u_{i2} = g^{t_{i2}})$ is the corresponding public key. The session key $s_i = (z_i = (\omega_i y_i^{h_{CL}(\mathtt{ID}_i, \omega_i)})^{r_i}, z_{i1} = u_{i1}^{r_i}, z_{i2} = u_{i2}^{r_i})$ and $\mathbb{G}' = \mathbb{G} \times \mathbb{G} \times \mathbb{G}$. The security analysis remains exactly the same except under the CDH assumption now. However, in the rest of the paper, we only consider the more efficient CL-OR protocol.

5 Performance Comparison

In this section, we compare the performance of the Sphinx-based circuit constructions of the three protocols with their original constructions. Specifically, we compare message sizes and computational cost.

5.1 Message Sizes

Along with the UC model security, message compactness is an important advantage of using Sphinx. It is easy to observe that the major savings in the length of a circuit construction message comes from reuse of a pseudonym to which blinding is added at each circuit node.

Following the Sphinx notation, p is the size of a public key element in group $\mathbb{G}$ and r is the maximum length of the circuit. We aim at $\kappa = 128$-bit security and use the elliptic curve (ECC) setting with points (compressed form) of size $p = 256$ bits, such as provided by Dan Bernstein's Curve 25519 [3] used by Sphinx. For the finite field setting ($\mathbb{F}$), as higher values amplify our advantage, we consider a DH modulus of size just $p = 2048$ bits to model 128-bit security. To mitigate a recent attack on Tor by Evans, Dingledine and Grothoff [17], which uses long circuit paths that loop back, the maximum circuit length for recent versions of Tor is set as 8. Therefore, we set $r = 8$ for our Sphinx-based design. However, while comparing, we give an advantage to the original Tor-preDH, PB-OR and CL-OR protocols by using Tor's default circuit size $\nu = 3$ for them; using $r = 3$ in our design will only increase our advantage. Additionally, see [33] for a discussion of the effect of Tor's "CREATE_FAST" mechanism.

In the Sphinx-based OR construction, the user sends the tuple $(\alpha_0, \beta_0, \gamma_0)$ to node n_0. The lengths of the elements in this tuple are p, $(2r-1)\kappa$ and κ respectively. The total length, therefore, is equal to $p + 2r\kappa$. In the chosen ECC setting, this is equal to 1280 bits, while for the chosen finite field setting, this is equal to 3072 bits. The message size does not depend upon a specific OR design.

In the original Tor-preDH, PB-OR and CL-OR protocols, this cost is equal to $r(p+2\kappa)$ as each layer of onion in those constructions requires p bits for a pseudonym, κ bits for identity of the nodes and κ bits for message integrity. With $\kappa = 128$ and $\nu = 3$, this length is equal to 1536 bits in the ECC setting and 6912 bits in the finite field setting. These values are significantly larger than those in our generic format that can make circuits of any length up to 8. Note that with the necessity of pairings in the PB-OR protocol, we do not consider the finite field setting for it.

Table 1: Comparison between lengths (in bits) of various single-pass OR circuit construction messages for 128-bit security ($\kappa = 128$)

Scheme	**Circuit Size**	**UC Security**	**Message Size (bits)**	$\mathbb{F}$ ($p = 2048$)	**ECC** ($p = 256$)
ØS07 [33]	$\nu = 3$	$\times$	$\nu(p + 2\kappa)$	6912	1536
PBOR [26]	$\nu = 3$	$\times$	$\nu(p + 2\kappa)$	–[a]	1536
CL-OR [9]	$\nu = 3$	$\times$	$\nu(p + 2\kappa)$	6912	1536
CL05 [7]	$\nu = 3$	$\surd$	$\nu(p + \kappa)$	6528	1920[b]
Sphinx-OR	$r = 8$	$\surd$	$\mathbf{p + 2r}\kappa$	**3072**	**1280**

[a] With the necessity of pairings in the PB-OR protocol, we do not consider the finite field setting for it. [b] As we use an Elgamal ciphertext in ECC, $p' = 2p = 512$.

We also consider Camenisch and Lysyanskaya's design in [7] that is secure in the UC model. The message length there is $r(p + \kappa)$, which is much larger than the Sphinx-based design. In the ECC computation, we use an Elgamal ciphertext of two $\mathbb{G}$ elements of length $p' = 2p = 512$ instead of p. For $\nu = 3$, the message sizes are 1920 bits and 6528 bits respectively. Therefore, our Sphinx-based design achieves the same security guarantees with much smaller messages. Table 1 provides a succinct representation of the above discussion. Note that as Tor generates a circuit in a telescoping form, we do not compare it with the single-pass protocols.

Although Sphinx achieves partial independence from the circuit size r with a single pseudonym, r is still present in the message-size expression due to node addresses and the integrity mechanism. In onion routing, a user should be able to choose its nodes from the available pool in an arbitrary fashion [39, 32], so information theoretically, it is impossible to make the message size independent of r.

5.2 Computational Cost

Compact messages and security in the UC model do not come without some additional computational cost. However, importantly, there is no addition to the computations done by users (possibly hundreds of thousands of them), while the increase is easily manageable for OR nodes.

The computation at a user end remains the same except for a few additional low-level operations such as a multiplication in $\mathbb{Z}_q$, a pseudorandom number generation and a few hashes having computational costs in μs. Each node in a circuit has to perform an additional exponentiation in $\mathbb{G}$ as it prepares the pseudonym for the next node. However, timing values computed using the pairing-based cryptography (PBC) library [28] indicate that one exponentiation in $\mathbb{G}$ costs around 1 ms on a desktop machine. This does not affect the overall circuit construction cost in practice, which is in seconds due to the network latency.

6 Comparison between Three OR Constructions

The three Sphinx-based OR circuit constructions that we present in §4 have been developed in three different settings; namely PKC, IBC and CLC. Although the papers presenting the original schemes compare their performance with Tor and also with each other to some extent, the analysis is not complete. In this section, we compare these three schemes in terms of their computational and infrastructural costs.

For all three schemes [33, 27, 9], their eventual forward secret constructions are significantly more efficient than the existing Tor circuit construction, which is obvious because of a linear message complexity gap between the two settings. The immediate forward secret constructions of Tor-preDH and PB-OR are also more efficient than Tor due to the high computational cost of the RSA decryptions. Immediate forward secrecy is not discussed for CL-OR [9]. Between Tor-preDH and PB-OR, the former is more efficient as it avoids pairings. However, Sphinx is more useful in the single-pass circuit constructions and we concentrate on the Sphinx-based versions of three single-pass OR schemes.

In Tor-preDH, to build a circuit of length ν, a user performs 2ν exponentiations in $\mathbb{G}$ (one for a pseudonym and other for a session key for each node), while each node performs two exponentiations in $\mathbb{G}$ (one for a pseudonym for the next node and other for the session key). In PB-OR, a user does ν exponentiations in $\mathbb{G}$ for pseudonyms and ν in $\mathbb{G}_T$ for session keys, while each node performs one exponentiation in $\mathbb{G}$ to compute a pseudonym for the next node and one pairing computation for the session key. In CL-OR, a user performs 3ν exponentiations in $\mathbb{G}$ (ν for pseudonyms and 2ν for session keys) and a node also performs three exponentiations in $\mathbb{G}$ (one for the pseudonym and two for the session key).

Although Tor-preDH is the most efficient circuit construction in terms of computations costs, from the systems perspective, there is a scalability challenge in it. Each OR node has to generate a DH pair, and self-sign and forward the public half to the directory server once every forward secrecy phase. In addition, the directory server has to verify and manage this signed public key list of all OR nodes and all users of the anonymity network (possibly hundreds of thousands of them) have to download the complete public key set as the new forward secrecy phase starts. Using an identity-based infrastructure, PB-OR completely avoids the above public key management (scalability) issue, but the *key escrow* available to the PKG presents an important privacy challenge. It is possible to mitigate this problem using a distributed PKG setup [24]. However, this not only asks for additional infrastructure, but also increases communication. Each OR node now has to communicate with at least a threshold number of PKGs to obtain its private key as the new forward secrecy phase starts. In CL-OR, the key escrow problem in the identity-based setting is mitigated by combining an identity-private key pair with a user-generated (uncertified) public-private key pair.[2] However, the public key management issue resurfaces here. Although not signed, each OR node has to generate a DH pair and forward the public half of it to the directory server per forward secrecy phase. The directory server has to maintain this (unsigned) public key list and all (possibly hundreds of thousands) of the users have to download the complete set of public keys as the new forward secrecy phase starts.

Apparently, none of the single-pass schemes is a clear winner. In the CLC setting, the problem could have been solved if it would have been possible to modify the identity-private key pair without modifying the public key for an OR node. In such a system, along with key escrow mitigation, a single PKG generates and delivers private keys to the nodes once during each forward secrecy phase. Importantly, users are not required to download any new public keys. However, it is not possible to modify CL-OR to achieve this property as its IBC part of the private key $z = k + h_{CL}(ID, \omega)s$ cannot be securely modified across forward secrecy phases without changing k and the associated public parameter $\omega = g^k$. Maintaining the same k across two phases makes it possible for any node to determine k and the master key s by solving a system of linear equations.

[2]The PKG can still apply an active (but detectable) attack to generate the DH pair itself.

7 Concluding Remarks and Future Work

Observing that the Sphinx message format defined for the mix networks is also applicable to OR circuit constructions, we designed a generic OR circuit construction, which is compact as well as secure in the UC model. Further, we used this generic construction to improve the circuit constructions for the Tor-preDH, PB-OR, CL-OR and 2-CL-OR protocols. From a practical perspective, we then compared the messages in the new circuit constructions with the original protocols and noted that the new messages are significantly smaller.

Moreover, we compared the three new schemes with Tor as well as with each other. We observed that in multi-pass constructions, Tor-preDH is the most efficient. However, in the absence of a clearly optimal scheme, the choice among the single-pass circuit constructions has to be made based on the size of a prospective anonymity network and availability of a PKG infrastructure. For smaller networks, Tor-preDH and CL-OR are better suited than PB-OR. However, the choice between those two is tricky. In Tor-preDH, the directory server and users have to verify OR nodes' public key certificates once per forward secrecy phase. In CL-OR, for every circuit construction a user has to perform ν additional exponentiations and every circuit node has to perform one additional exponentiation. For large anonymity networks, we find PB-OR to be more usable. The public-key downloads saved there are more than compensate for the infrastructure cost incurred by a (distributed) PKG.

Further, using the CLC setting, it may be possible to avoid the public-key scalability and key escrow issues at the same time and it is an interesting future work to design such a scheme. Finally, all of the above schemes assume random oracles and it would also be interesting to define an OR circuit construction which is secure without random oracles.

Acknowledgements. We thank D. Fiore for providing the camera-ready version of his certificateless onion routing paper [9] with D. Catalano and R. Gennaro. We also thank R. Dingledine, G. Zaverucha, and the anonymous reviewers for providing valuable feedback. This work is supported by NSERC, MITACS, and a David R. Cheriton Graduate Scholarship.

References

[1] M. Abdalla, M. Bellare, and P. Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In *CT-RSA'01*, pages 143–158, 2001.

[2] S. S. Al-Riyami and K. G. Paterson. Certificateless Public Key Cryptography. In *Advances in Cryptology—ASIACRYPT'03*, pages 452–473, 2003.

[3] D. J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In *Public Key Cryptography (PKC'06)*, pages 207–228, 2006.

[4] I. Blake, G. Seroussi, and N. P. Smart, editors. *Advances in Elliptic Curve Cryptography.* Number 317 in London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge, UK, 2005. 183–252.

[5] D. Boneh. The Decision Diffie-Hellman Problem. In *Third International Symposium Algorithmic Number Theory (ANTS-III)*, pages 48–63, 1998.

[6] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology—CRYPTO'01*, pages 213–229, 2001.

[7] J. Camenisch and A. Lysyanskaya. A Formal Treatment of Onion Routing. In *Advances in Cryptology—CRYPTO'05*, pages 169–187, 2005.

[8] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS'01*, pages 136–145, 2001.

[9] D. Catalano, D Fiore, and R. Gennaro. Certificateless Onion Routing. In *CCS'09*, pages 151–160, 2009.

[10] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 4(2):84–88, 1981.

[11] W. Dai. PipeNet 1.1. `http://www.weidai.com/pipenet.txt`, 1998. Accessed Nov. 2009.

[12] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *IEEE Symposium on Security and Privacy*, pages 2–15, 2003.

[13] G. Danezis and I. Goldberg. Sphinx: A Compact and Provably Secure Mix Format. In *IEEE Symposium on Security and Privacy*, pages 269–282, 2009.

[14] G. Danezis and B. Laurie. Minx: A Simple and Efficient Anonymous Packet Format. In *WPES'04*, pages 59–65, 2004.

[15] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transaction on Information Theory*, 22(6):644–654, 1976.

[16] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium*, pages 303–320, 2004.

[17] N. S. Evans, R. Dingledine, and C. Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *18th USENIX Security Symposium*, pages 33–50, 2009.

[18] M. J. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *CCS'02*, pages 193–206. ACM, 2002.

[19] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for Cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[20] I. Goldberg. On the Security of the Tor Authentication Protocol. In *PET'06*, pages 316–331, June 2006.

[21] D. M. Goldschlag, M. Reed, and P. Syverson. Hiding Routing Information. In *Information Hiding: First International Workshop*, pages 137–150, 1996.

[22] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion Routing. *Commun. ACM*, 42(2):39–41, 1999.

[23] A. Joux. The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems. In *ANTS-V*, pages 20–32, 2002.

[24] A. Kate and I. Goldberg. Asynchronous Distributed Private-Key Generators for Identity-Based Cryptography. Cryptology ePrint Archive, Report 2009/355, 2009.

[25] A. Kate and I. Goldberg. Using Sphinx to Improve Onion Routing Circuit Construction. In *FC'10*, 2010.

[26] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing. In *PETS'07*, pages 95–112, 2007.

[27] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing with Improved Forward Secrecy. *To appear in ACM TISSec*, 2009.

[28] B. Lynn. PBC Library – The Pairing-Based Cryptography Library. `http://crypto.stanford.edu/pbc/`, 2006. Accessed September 2009.

[29] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1st edition, 1997.

[30] B. Möller. Provably Secure Public-Key Encryption for Length-Preserving Chaumian Mixes. In *CT-RSA'03*, pages 244–262, 2003.

[31] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol— Version 2. IETF Internet Draft, 2003.

[32] S. J. Murdoch and R. N. M. Watson. Metrics for Security and Performance in Low-Latency Anonymity Systems. In *PETS'08*, pages 115–132, 2008.

[33] L. Øverlier and P. Syverson. Improving Efficiency and Simplicity of Tor Circuit Establishment and Hidden Services. In *PETS'07*, pages 134–152, 2007.

[34] J. Reardon and I. Goldberg. Improving Tor Using a TCP-over-DTLS Tunnel. In *18th USENIX Security Symposium*, pages 119–133, 2009.

[35] M. Reed, P. Syverson, and D. Goldschlag. Anonymous Connections and Onion Routing. *IEEE J-SAC*, 16(4):482–494, 1998.

[36] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *WPES'02*, pages 91–102, 2002.

[37] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems Based on Pairing. In *Symposium on Cryptography and Information Security (SCIS'00)*, Japan, 2000.

[38] E. Shimshock, M. Staats, and N. Hopper. Breaking and Provably Fixing Minx. In *PETS'08*, pages 99–114, 2008.

[39] R. Snader and N. Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *NDSS'08*, 2008.

[40] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In *Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114, 2000.

[41] The Tor Project. . `https://www.torproject.org/`, 2003. Accessed Nov. 2009.

A Bilinear Pairings

For three cyclic groups $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ (all of which we shall write multiplicatively) of the same prime order p, a *bilinear pairing* e is a map $e : \mathbb{G} \times \hat{\mathbb{G}} \rightarrow \mathbb{G}_T$ with the following properties.

- **Bilinearity:** For $g \in \mathbb{G}$, $\hat{g} \in \hat{\mathbb{G}}$ and $a, b \in \mathbb{Z}_p$, $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab}$.
- **Non-degeneracy:** The map does not send all pairs in $\mathbb{G} \times \hat{\mathbb{G}}$ to unity $\in \mathbb{G}_T$.

If there is an efficient algorithm to compute $e(g, \hat{g})$ for any $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$, the pairing e is called *admissible*. We also expect that it is not feasible to invert a pairing and come back to $\mathbb{G}$ or $\hat{\mathbb{G}}$. All pairings considered in this paper are admissible and infeasible to invert. We call such groups $\mathbb{G}$ and $\hat{\mathbb{G}}$ *pairing-friendly* groups. We refer readers to [4, Chap. IX and X] for a detailed mathematical discussion of bilinear pairings.

Following [19], we consider three types of pairings: namely, type 1, 2, and 3. In *type* 1 pairings, an isomorphism $\phi : \hat{\mathbb{G}} \rightarrow \mathbb{G}$ as well as its inverse ϕ^{-1} are efficiently computable. These are also called *symmetric pairings* as for such pairings $e(g, \hat{g}) = e(\phi(\hat{g}), \phi^{-1}(g))$ for any $g \in \mathbb{G}$ and $\hat{g} \in \hat{\mathbb{G}}$. In *type* 2 pairings, only the isomorphism ϕ, but not ϕ^{-1}, is efficiently computable. Finally in *type* 3 pairings, neither of ϕ nor ϕ^{-1} can be efficiently computed. The efficiency of the pairing computation improves from type 1 to type 2 to type 3 pairings. For a detailed discussion of the performance aspects of pairings we refer the reader to a survey by Galbraith et al. [19].