Pairing-Based Onion Routing with Improved Forward Secrecy [*]

Aniket Kate   Greg M. Zaverucha   Ian Goldberg

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada N2L 3G1
{akate,gzaveruc,iang}@cs.uwaterloo.ca

Abstract

This paper presents new protocols for onion routing anonymity networks. We define a provably secure privacy-preserving key agreement scheme in an identity-based infrastructure setting, and use it to forge new onion routing circuit constructions. These constructions, based on a user's selection, offer immediate or eventual forward secrecy at each node in a circuit and require significantly less computation and communication than the telescoping mechanism used by Tor. Further, the use of the identity-based infrastructure also leads to a reduction in the required amount of authenticated directory information. Therefore, our constructions provide practical ways to allow onion routing anonymity networks to scale gracefully.

Keywords: onion routing, Tor, pairing-based cryptography, anonymous key agreement, forward secrecy

1 Introduction

Over the years, a large number of anonymity networks have been proposed and some have been implemented. Common to many of them is *onion routing* [RSG98], a technique whereby a message is wrapped in multiple layers of encryption, forming an *onion*. As the message is delivered via a number of intermediate *onion routers* (ORs), or *nodes*, each node decrypts one of the layers, and forwards the message to the next node. This idea goes back to Chaum [Cha81] and has been used to build both low- and high-latency communication networks.

A common realization of an onion routing system is to arrange a collection of nodes that will relay traffic for users of the system. Users then randomly choose a path through the network of onion routers and construct a *circuit*—a sequence of nodes which will route traffic. After the circuit is constructed, each of the nodes in the circuit shares a symmetric key with the user, which will be used to encrypt the layers of future onions.

Pairing-based cryptography (see [KM05] for a detailed discussion) has drawn an overwhelming amount of research attention in the last few years. In one of the pioneering works in the field, Sakai, Ohgishi, and Kasahara [SOK00] presented a non-interactive key agreement scheme in the identity-based setting. In this article, we enhance their protocol to develop provably secure one- or two-way (also referred to as unilateral or bilateral) privacy-preserving authentication and key agreement schemes. After one-way authentication between Anonymous and Bob, Anonymous has

[*]This is an extended version of our paper "Pairing-Based Onion Routing", which appeared in the proceedings of the 7th Privacy Enhancing Technologies Symposium (LNCS vol. 4776).

confirmed Bob's identity and Bob learns nothing about Anonymous, except perhaps that he or she is a valid user of a particular system. In a two-way scheme, each user can confirm the other is a valid user without learning who the other is.

We then use our one-way anonymous key agreement protocol to build onion routing circuits for anonymity networks like Tor [DMS04] and prove security-related properties of the new construction. Our protocol, which first appeared in [KZG07], constructs a circuit in a single pass and also provides a practical way to achieve eventual forward secrecy. Observing the performance trade-off between immediate and eventual forward secrecy in onion routing circuit construction, we also develop a λ-pass circuit construction, which obtains immediate forward secrecy at λ nodes. The performance of our circuit construction protocols surpass that of Tor, requiring significantly less computation and fewer network communications. Further, they do not require the public keys of onion routers to be authenticated and consequently, reduce the load on directory servers and improves the scalability of anonymity networks.

Previous work related to pairing-based key exchange, as well as to anonymity networks, is covered in Section 2. We describe the cryptographic protocols in Section 3, and an onion routing system built with a Boneh-Franklin identity-based infrastructure in Section 4. In Section 5, we present our λ-pass circuit construction and discuss the security of our onion routing circuit constructions in Section 6. Some of the more practical issues in such a system are discussed in Section 7 and we compare our computational and communication costs to those of Tor in Section 8.

2 Related Work

The concept of onion routing plays a key role in many efforts to provide anonymous communication [Dai98, DMS04, FM02, RSG98, RP02] while a number of other papers discuss formalizations and the security of onion routing [CL05, MVdV04, Möl03, STRL00]. To date, the largest onion routing system is Tor, which has approximately 1000 onion routers and hundreds of thousands of users [The08]. These numbers (and their growth) underscore the demand for anonymity online.

In the original Onion Routing project [GRS96, RSG98, STRL00] (which was superseded by Tor) circuit construction was done as follows. The user created an onion where each layer contained the symmetric key for one node and the location of the next node, all encrypted with the original node's public key. Each node decrypts a layer, keeps the symmetric key and forwards the rest of the onion along to the next node. The main drawback of this approach is that it does not provide forward secrecy (as defined in [DMS04]). Suppose a circuit is constructed from the user to the sequence of nodes $A \Leftrightarrow B \Leftrightarrow C$, and that A is malicious. If A records the traffic, and at an arbitrary time in the future compromises B (at which point he learns the next hop is C), and then compromises C, the complete route is known, and A learns who the user has communicated with. A possible fix for this problem is to frequently change the public keys of each node. This limits the amount of time A has to compromise B and C, but requires that the users of the system frequently contact the directory server to retrieve authentic keys.

Later systems constructed circuits incrementally and interactively (this process is sometimes called *telescoping*). The idea is to use the node's public key only to initiate a communication during which a temporary session key is established via the Diffie-Hellman key exchange. Tor constructs circuits in this way, using the Tor authentication protocol (TAP). TAP is described and proven secure in previous work of the last author [Gol06].

Trade-offs exist between the two methods of constructing circuits. Forward secrecy is the main advantage of telescoping, but telescoping also handles nodes that are not accepting connections; if the third node is down during the construction of a circuit, for example, the first two remain, and the user only needs to choose an alternate third. Information about the status and availability of nodes is therefore less important. The drawback of telescoping is the cost; establishing a circuit of length ℓ requires $O(\ell^2)$ network communications, and $O(\ell^2)$ symmetric encryptions/decryptions.

Øverlier and Syverson [ØS07] improve the efficiency of telescoping-based circuit construction using a half-certified Diffie-Hellman key exchange [MOV97, Sec. 12.6]. They further define an efficient single-pass circuit construction and a few variants. The proposed variants offer different levels of forward secrecy, which is traded off against computation and communication. For example, their eventual forward secret variants use frequent rotation of nodes' public keys, presenting the same issues as in first-generation onion routing; their immediate forward secrecy variant uses the same amount of communication as the current Tor ($O(\ell^2)$), but less computation.

In related efforts, Camenisch and Lysyanskaya [CL05] formally define the requirements of a secure onion routing construction in the universal composability (UC) framework [Can01] and present a generic construction of onion routing circuits. Although formally secure, their construction is less efficientthan other constructions due to the additional mechanisms required to prove security in the UC framework. While no attacks are known when these mechanisms are removed, the proof no longer holds.

The work of Okamoto and Okamoto [OO05] presents schemes for anonymous authentication and key agreement. In Rahman et. al. [RIO$^+$06], an anonymous authentication protocol is presented as part of an anonymous communication system for mobile ad-hoc networks. The protocols in both papers are complex, and limited motivation is given for design choices. Further, both papers neglect to discuss the security of their proposed protocols. The protocols we present in Section 3.2 are a great deal simpler than previous protocols. This allows them to be more easily understood, and simplifies the discussion of their security, which appears in Section 3.3. A recent article [Hua07] presents a pseudonym-based encryption scheme similar to our anonymous key protocol in Section 3.2, but differs in its method of private-key extraction as well as in the motivation behind its use.

All these protocols owe a lot to the non-interactive key exchange protocol of Sakai, Ohgishi and Kasahara [SOK00]. In the next section, we will review their scheme after covering relevant background material.

3 Pairing-Based Key Agreement with User Anonymity

In one of the pioneering works of pairing-based cryptography, Sakai et al. suggested an identity-based non-interactive key agreement scheme using bilinear pairings [SOK00]. In this section, we extend this key agreement scheme. We replace the identities of the participants by pseudonyms and our new scheme provides unconditional anonymity to participating users.

3.1 Preliminaries

We briefly review bilinear pairings and the original non-interactive key agreement scheme of Sakai et al. For a detailed presentation of pairings and cryptographic applications thereof see Blake et al. [Bla05] and references therein.

3.1.1 Bilinear Pairings

Consider two additive cyclic groups $\mathbb{G}$ and $\hat{\mathbb{G}}$ and a multiplicative cyclic group $\mathbb{G}_T$, all of the same prime order n. A bilinear map e is a map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ with following properties.

1. **Bilinearity:** For all $P \in \mathbb{G}$, $Q \in \hat{\mathbb{G}}$ and $a, b \in \mathbb{Z}_n$, $e(aP, bQ) = e(P, Q)^{ab}$.

2. **Non-degeneracy:** The map does not send all pairs in $\mathbb{G} \times \hat{\mathbb{G}}$ to unity in $\mathbb{G}_T$.
3. **Computability:** There is an efficient algorithm to compute $e(P, Q)$ for any $P \in \mathbb{G}$ and $Q \in \hat{\mathbb{G}}$.

Our protocols, like many pairing-based cryptographic protocols, use a special form of bilinear map called a *symmetric pairing* which has $\mathbb{G} = \hat{\mathbb{G}}$. For such pairings $e(P, Q) = e(Q, P)$ for any $P, Q \in \mathbb{G}$. The modified Weil pairing over elliptic curve groups [Ver01] is an example of a symmetric bilinear pairing. In the rest of the paper, unless otherwise specified, all bilinear pairings are symmetric.

3.1.2 The Bilinear Diffie-Hellman Assumption

With a groups $\mathbb{G}$ and $\mathbb{G}_T$ and a pairing e of the above type, the *Bilinear Diffie-Hellman* (BDH) problem is to compute $e(P, P)^{abc} \in \mathbb{G}_T$ given a generator P of $\mathbb{G}$ and elements aP, bP, cP for $a, b, c \in \mathbb{Z}_n^*$. An equivalent formulation of the problem, due to the bilinearity of the map, is to compute $e(A, B)^c$ given a generator P of $\mathbb{G}$, and elements A, B and cP in $\mathbb{G}$.

If there is no efficient algorithm to solve the BDH problem for $\langle \mathbb{G}, \mathbb{G}_T, e \rangle$, they are said to satisfy the *BDH assumption*.

3.1.3 Boneh-Franklin Setup and Non-Interactive Key Agreement

In a Boneh-Franklin Identity-Based Encryption (BF-IBE) setup [BF01], a trusted authority, called a private key generator (PKG), generates private keys (d_i) for clients using the clients' well-known identities ($\mathtt{ID}_i$) and a master secret s. A client with identity $\mathtt{ID}_i$ receives the private key $d_i = sH(\mathtt{ID}_i) \in \mathbb{G}$, where $H : \{0,1\}^* \rightarrow \mathbb{G}^*$ is a full-domain cryptographic hash function and $\mathbb{G}^*$ denotes the set of all elements in $\mathbb{G}$ except the identity.

Sakai et al. observed that, with such a setup, any two clients of the same PKG can compute a shared key using only the identity of the other participant and their own private keys. Only the two clients and the PKG can compute this key. For two clients with identities $\mathtt{ID}_A$ and $\mathtt{ID}_B$, the shared key is given by $K_{AB} = e(Q_A, Q_B)^s = e(Q_A, d_B) = e(d_A, Q_B)$ where $Q_A = H(\mathtt{ID}_A)$ and $Q_B = H(\mathtt{ID}_B)$.

Dupont and Enge proved this protocol is secure in the random oracle model assuming the BDH problem in $\langle \mathbb{G}, \mathbb{G}_T, e \rangle$ is hard [DE06].

3.2 Anonymous Key Agreement

We observe that by replacing the identity hashes with pseudonyms generated by users, a key agreement protocol with unconditional anonymity is possible. In our protocol, each participant can confirm that the other participant is a client of the same PKG, but cannot determine his identity. Each client can, on her own, randomly generate many possible pseudonyms and the corresponding private keys.

Suppose Alice, with (identity, private key) pair $(\mathtt{ID}_A, d_A)$, is seeking anonymity. She generates a random number $r_A \in \mathbb{Z}_n^*$ and creates the pseudonym and corresponding private key $(P_A = r_A Q_A = r_A H(\mathtt{ID}_A), r_A d_A = sP_A)$. In a key agreement protocol, she sends the pseudonym P_A instead of her actual identity to another participating client, who may or may not be anonymous. For two participants (say Alice and Bob) with pseudonyms P_A and P_B, the shared session key is given as

$$K_{AB} = e(P_A, P_B)^s = e(Q_A, Q_B)^{r_A r_B s}$$

where r_A and r_B are random numbers generated respectively by Alice and Bob. If Bob does not wish to be anonymous, he can just use $r_B = 1$ instead of a random value, resulting in $P_B = Q_B$. If persistent pseudonymity is desired instead of anonymity, the random values can easily be reused.

Two participants can perform a session key agreement by exchanging pseudonyms. Further, two participants can also perform an authenticated key agreement by modifying any secure symmetric-key based mutual authentication protocol and simply replacing their identities by their pseudonyms.

3.2.1 One-Way Anonymous Key Agreement

Anonymous communication often requires anonymity for just one of the participants; the other participant works as a service provider and the anonymous participant needs to confirm her identity. In the key agreement protocol, the service provider uses her actual identity rather than a pseudonym. Further, in this one-way anonymity setting two participants can agree on a session key in a non-interactive manner. A non-interactive scheme to achieve this is defined next.

Suppose Alice and Bob are clients of a PKG. As before, Alice has identity $\mathtt{ID}_A$ and private key $d_A = sQ_A = sH(\mathtt{ID}_A)$. Alice wishes to remain anonymous to Bob, but she knows Bob's identity $\mathtt{ID}_B$.

1. Alice computes $Q_B = H(\mathtt{ID}_B)$. She chooses a random integer $r_A \in \mathbb{Z}_n^*$, generates the corresponding pseudonym $P_A = r_A Q_A$ and private key $r_A d_A = sP_A$, and computes the session key $K_{AB} = e(sP_A, Q_B) = e(Q_A, Q_B)^{sr_A}$. She sends her pseudonym P_A to Bob.
2. Bob, using P_A and his private key d_B, computes the session key $K_{AB} = e(P_A, d_B) = e(Q_A, Q_B)^{sr_A}$.

Note that in step 1, Alice can also include a message for Bob symmetrically encrypted with the session key; we will use this in Section 4. Note also that in practice, the session key is often derived from K_{AB}, and is not just K_{AB} itself.

3.2.2 Key Authentication and Confirmation

In most one-way anonymous communication situations, it is also required to authenticate the non-anonymous service provider. With the non-interactive protocols of this section, the key is implicitly authenticated; Alice is assured that only Bob can compute the key. If Alice must be sure Bob has in fact computed the key, explicit key confirmation can be achieved by incorporating any symmetric-key based challenge-response protocol.

3.3 Security and Anonymity

In this section, we discuss the security and anonymity of our key agreement schemes in the random oracle model. We make the following claims:

Unconditional Anonymity It is impossible for the other participant in a protocol run, the PKG, or any third party to learn the identity of an anonymous participant in a protocol run.

Session Key Secrecy It is infeasible for anyone other than the two participants or the PKG to determine a session key generated during a protocol run.

No Impersonation It is infeasible for a malicious client of the PKG to impersonate another (non-anonymous) client in a protocol run. In the case of persistent pseudonymity, it is not feasible for a malicious entity to communicate using a different entity's pseudonym.

Next, we prove each of our claims.

3.3.1 Unconditional Anonymity

Here, we prove that it is impossible for an adversary $\mathcal{A}$ to learn the identity of an anonymous participant in a protocol run. Informally, for an anonymous client with identity $\texttt{ID}_A$, the pseudonym $P_A = r_A Q_A \in \mathbb{G}$ is the only parameter exchanged during the protocol that is derived from her identity. Because $\mathbb{G}$ is a cyclic group of prime order, Q_A is a generator, so multiplying by the random r_A perfectly blinds the underlying identity from the adversary $\mathcal{A}$, which can be the other participant in the protocol run, the PKG for the system or any third party. To formalize our proof, we consider the following game between an adversary and a challenger.

Setup The adversary $\mathcal{A}$ publishes the system parameters: a cyclic additive group $\mathbb{G}$ of prime order n and a hash function $H : \{0,1\}^* \to \mathbb{G}^*$.

Challenge $\mathcal{A}$ chooses two identity strings $\texttt{ID}_A$ and $\texttt{ID}_B$ and sends them to the challenger. The challenger computes $Q_A = H(\texttt{ID}_A)$ and $Q_B = H(\texttt{ID}_B)$. He then uniformly at random chooses $r \in \mathbb{Z}_n^*$ and $b \in \{0,1\}$, then

1. if $b = 0$, computes a pseudonym $P = rQ_A$ or
2. if $b = 1$, computes a pseudonym $P = rQ_B$

and sends P to $\mathcal{A}$.

Guess $\mathcal{A}$ wins the game if she can guess the correct value of b with probability significantly greater than 1/2.

As $\mathbb{G}$ is a cyclic prime order group, both Q_A and Q_B are generators of $\mathbb{G}$. For the uniform random element $r \in \mathbb{Z}_n^*$, the pseudonym P equal to rQ_A or rQ_B is also a uniform random element of $\mathbb{G}^*$. Therefore, an attacker cannot determine which of the two ways the challenger generated P and consequently cannot guess the value of b with probability greater than 1/2 to win this game. The inability of the attacker to win this game for system parameters of their choosing, even with unbounded computation power, proves our unconditional anonymity claim.

3.3.2 Session Key Secrecy

Dupont and Enge [DE06] prove the security of the key agreement scheme of Sakai et al. in the random oracle model. According to their proof, an attacker cannot compute the shared key if the BDH assumption holds on $\langle \mathbb{G}, \mathbb{G}_T, e\rangle$, and H is modelled by a random oracle. Here, we modify their proof to prove that it is infeasible for anyone other than the two participants or the PKG to determine a session key generated during a protocol run of the one-way or two-way anonymous key agreement.

Consider the following game to prove key secrecy in the one-way anonymous case.

Setup The challenger generates groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order n, a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}^*$, a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_T$ and a master secret $s \in \mathbb{Z}_n^*$.

Extraction Queries The adversary $\mathcal{A}_1$ issues q extraction queries for identities $\texttt{ID}_1, \texttt{ID}_2, \ldots, \texttt{ID}_q$ to the challenger. The challenger queries H to compute the corresponding private keys $sH(\texttt{ID}_1), sH(\texttt{ID}_2), \ldots, sH(\texttt{ID}_q)$ and sends them back to $\mathcal{A}_1$.

Challenge Once $\mathcal{A}_1$ informs the challenger that it has collected enough information, the challenger picks an element $P_A \in \mathbb{G}^*$ and sends it to $\mathcal{A}_1$.

Guess $\mathcal{A}_1$ outputs a binary string (an identity) $\mathtt{ID}_B$ and $K_{AB} \in \mathbb{G}_T$.

The attacker's advantage can be defined as

$$\text{Adv}(\mathcal{A}_1) = \text{Prob}(e(P_A, H(\mathtt{ID}_B))^s = K_{AB})$$

We say $\mathcal{A}_1$ (t_1, ϵ_1)-wins the game, if it runs in time at most t_1 and has advantage ϵ_1.

Next, consider the following game to prove key secrecy in the two-way anonymous case.

Setup The challenger generates groups $\mathbb{G}$ and $\mathbb{G}_T$ of order n, a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}^*$, a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_T$ and a master secret $s \in \mathbb{Z}_n^*$.

Extraction Queries The adversary $\mathcal{A}_2$ issues q extraction queries for identities $\mathtt{ID}_1, \mathtt{ID}_2, \ldots, \mathtt{ID}_q \in \mathbb{G}$ to the challenger. The challenger queries H to compute the corresponding private keys $sH(\mathtt{ID}_1), sH(\mathtt{ID}_2), \ldots, sH(\mathtt{ID}_q)$ and sends them back to $\mathcal{A}_2$.

Challenge Once $\mathcal{A}_2$ informs the challenger that it has collected enough information, the challenger picks two elements P_A and P_B in $\mathbb{G}^*$ and sends them to $\mathcal{A}_2$.

Guess $\mathcal{A}_2$ outputs $K_{AB} \in \mathbb{G}_T$.

The attacker's advantage can be defined as

$$\text{Adv}(\mathcal{A}_2) = \text{Prob}(e(P_A, P_B)^s = K_{AB})$$

We say $\mathcal{A}_2$ (t_2, ϵ_2)-wins a game, if it runs in time at most t_2 and has advantage ϵ_2.

Suppose that there is an adversary $\mathcal{A}_1$ who (t_1, ϵ_1)-wins the one-way anonymous game and an adversary $\mathcal{A}_2$ who (t_2, ϵ_2)-wins the two-way anonymous game. We now show that an algorithm $\mathcal{B}$ can make use of $\mathcal{A}_1$ or $\mathcal{A}_2$ to solve a random instance of the BDH problem.

Theorem 3.1. *Let the hash function H be modelled by a random oracle. Suppose there exist adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ such that the adversary $\mathcal{A}_1$ (t_1, ϵ_1)-wins the one-way anonymous protocol security game and the adversary $\mathcal{A}_2$ (t_2, ϵ_2)-wins the two-way anonymous protocol security game. Then there exists an algorithm $\mathcal{B}$ which solves the BDH problem*

- *using the adversary $\mathcal{A}_1$ with probability $\frac{\epsilon_1}{e(1+q)}$ in time $t_1 + wq + t_T + t_{inv}$ or*
- *using the adversary $\mathcal{A}_2$ with probability ϵ_2 in time $t_2 + wq$.*

Here e is the base of natural logarithms, w is a small constant, q is an upper bound on the number of extraction queries performed by an adversary, t_T is the time required for exponentiation in $\mathbb{G}_T$ and t_{inv} is the time required to invert an element of $\mathbb{Z}_n^*$.

Proof. Let $(P, aP, bP, cP) \in \mathbb{G}$ be a random and uniformly distributed instance of the BDH problem, which algorithm $\mathcal{B}$ receives as input. To find the solution $e(P,P)^{abc}$, $\mathcal{B}$ simulates the challenger for $\mathcal{A}_1$ or $\mathcal{A}_2$. This means that $\mathcal{B}$ must simulate the random oracle H and answer the private key extraction queries by $\mathcal{A}_1$ or $\mathcal{A}_2$. As the steps for H-queries and extraction queries are the same in both $\mathcal{A}_1$ or $\mathcal{A}_2$, we denote both of them by $\mathcal{A}$.

H-queries At any time, $\mathcal{A}$ can query the random oracle H. To respond to these queries, $\mathcal{B}$ maintains an initially empty list L of tuples $(X, Q, h, \beta) \in \{0,1\}^* \times \mathbb{G}^* \times \mathbb{Z}_n^* \times \{0,1\}$. When $\mathcal{A}$ queries for the hash value of some bit-string X_i, algorithm $\mathcal{B}$ responds as follows:

1. If L contains a tuple (X_i, Q_i, h_i, β_i), $\mathcal{B}$ responds by sending Q_i.
2. Otherwise, $\mathcal{B}$ generates at random $\beta_i \in \{0,1\}$, so that $\text{Prob}(\beta_i = 0) = \delta$, where δ depends on B's choice for the attacker ($\mathcal{A}_1$ or $\mathcal{A}_2$) and will be determined below.
3. Algorithm $\mathcal{B}$ picks a random $h_i \in \mathbb{Z}_n^*$. If $\beta_i = 0$, set $Q_i = h_i P$, else set $Q_i = h_i(bP)$. Note that either way, Q_i is uniformly random in $\mathbb{G}^*$ and independent of $\mathcal{A}$'s current view.
4. Finally, algorithm $\mathcal{B}$ adds the tuple (X_i, Q_i, h_i, β_i) to the list L and responds with Q_i.

Extraction queries $\mathcal{A}$ can ask for extraction queries for identity strings. For an input string $\mathtt{ID}_i$ for private key extraction, $\mathcal{B}$ responds as follows:

1. Algorithm $\mathcal{B}$ runs the above H-query algorithm for input $X_i = \mathtt{ID}_i$ to obtain $(\mathtt{ID}_i, Q_i, h_i, \beta_i)$.
2. If $\beta_i = 1$ then $\mathcal{B}$ reports failure.
3. Otherwise, $\mathcal{B}$ computes the private key $h_i(cP) = cQ_i$ and sends it to algorithm $\mathcal{A}$.

Challenge After completing the extraction queries, B challenges $\mathcal{A}_1$ with $P_A = aP$ or $\mathcal{A}_2$ with $P_A = aP$ and $P_B = bP$.

Guess $\mathcal{A}_1$ outputs $(\mathtt{ID}_B, K_{AB}) \in \{0,1\}^* \times \mathbb{G}_T$ or A_2 outputs $K_{AB} \in \mathbb{G}_T$. In the case of adversary $\mathcal{A}_2$, $\mathcal{B}$ outputs $\sigma = K_{AB}$ as its guess for the solution to the BDH problem. For the adversary $\mathcal{A}_1$, algorithm $\mathcal{B}$ performs following steps:

1. $\mathcal{B}$ obtains the tuple $(ID_B, Q_B, h_B, \beta_B)$ from the list L. Absence of the tuple $(ID_B, Q_B, h_B, \beta_B)$ in the list L indicates that $\mathcal{A}_1$ did not ask the random oracle for $H(\mathtt{ID}_B)$. As the probability of the adversary's success in this case is negligible[1], we safely assume the presence of the tuple (ID_B, Q_B, h_B, c_B).
2. If $\beta_B = 1$, $\mathcal{B}$ outputs $\sigma = K_{AB}^{h_B^{-1}}$ as its guess for the BDH instance.
3. If $\beta_B = 0$, $\mathcal{B}$ reports failure.

Suppose that $\mathcal{B}$ does not report failure and outputs σ while using $\mathcal{A}_1$. As $\beta_B = 1$, $H(ID_B) = h_B(bP)$ and with probability ϵ_1, $\sigma = e(aP, h_B(bP))^{ch_B^{-1}} = e(P,P)^{abc}$. Therefore $\mathcal{B}$ will guess correctly with probability ϵ_1, when it does not abort. The probability that $\mathcal{B}$ does not abort while extracting a single private key query is δ; for q queries, the probability is δ^q. The probability that $\mathcal{B}$ does not abort while guessing the BDH solution is $1 - \delta$. Therefore, the overall probability of non-abortion is $\delta^q(1-\delta)$. Maximizing this probability, the optimal value can be obtained at $\delta = \frac{q}{1+q}$ and by choosing the value of δ optimally, the overall probability of non-abortion is $\frac{q^q}{(1+q)^{q+1}}$. Therefore, $\mathcal{B}$ outputs the correct solution to the BDH instance with probability at least $\frac{\epsilon_1 q^q}{(1+q)^{q+1}} \geq \frac{1}{e(1+q)}$ as $(1 - \frac{1}{q+1})^q \geq 1/e$. The solution is computed in time $t_1 + wq + t_T + t_{inv}$, where t_1 is the time required by $\mathcal{A}$, w is the time required to answer an extraction query (generate a random element r and compute the r-th multiple of cP), q is an upper bound on the number of such queries, and $t_T + t_{inv}$ is the time to invert an element of $\mathbb{Z}_n^*$ and to compute an exponentiation of $K_{AB} \in \mathbb{G}_T$ in the guessing phase.

For adversary $\mathcal{A}_2$, we simply set the value of δ to 1. Suppose that $\mathcal{B}$ does not report failure and outputs σ while using $\mathcal{A}_2$. With probability ϵ_2, $\sigma = e(aP, bP)^c = e(P,P)^{abc}$, which is the correct solution to the BDH problem. The solution is computed in time $t_2 + wq$. □

[1] If $\mathcal{A}_1$ does not query the random oracle for $H(\mathtt{ID}_B)$, the probability it can guess this value is negligible. As there is a bijection between x and $e(P_A, x)$ for a given P_A, the probability that $\mathcal{A}_1$ can output $K_{AB} = e(P_A, H(\mathtt{ID}_B))$ is also negligible. Thus, $\mathcal{A}_1$'s advantage in this case is negligible.

Note that it is possible to prove the security of the two-way anonymous key agreement protocol without random oracles, if we do not consider the query extraction phase. Assume that only one identity hash and private key pair (U, sU) is publicly available and each user uses the same pair to generate a pseudonym and corresponding private key. Given an adversary $\mathcal{A}$ to (t, ϵ)-compute $K_{AB} = e(P_A, P_B)^s$ when challenged by P_A and P_B, a random instance (P, aP, bP, cP) of the BDH problem can be solved in time t with probability ϵ by publishing (P, cP) as the publicly available identity hash and private key and challenging $\mathcal{A}$ with $P_A = aP$ and $P_B = bP$.

3.3.3 No Impersonation

We claim that it is infeasible for a malicious client of the PKG to impersonate another (non-anonymous) client in a protocol run. To successfully impersonate a non-anonymous participant ID_N in our one-way anonymous key agreement protocol, given a pseudonym and ID_N, an adversary needs to determine the corresponding session key. We observe that the adversary game for non-anonymous participant impersonation is the same as the key secrecy game of the one-way anonymous key agreement in Section 3.3.2 and consequently the corresponding theorem and proof carry over.

In the case of persistent pseudonymity, we claim that it is not feasible for a malicious entity to communicate using a different entity's pseudonym. Here, the malicious entity needs to find the shared secret key for a persistent pseudonym generated and used by some other anonymous entity and an arbitrary identity or pseudonym for which it does not know the private key. In the one-way anonymous communication protocol, the corresponding adversary game remains the same as that for impersonation of the non-anonymous entity, and in the two-way anonymous case, the game is the same as the one used to prove key secrecy. Consequently, the theorem and proof for the corresponding game are same as those used to prove key secrecy in Section 3.3.2.

3.4 Distributed PKG

The PKG in the BF-IBE framework, with the master key, has the power to decrypt all messages encrypted for clients. As our schemes use the same setup as BF-IBE, the PKG can compute a session key from the publicly available pseudonyms and the master key s. Due to this, compromise of the PKG is a single point of failure for security.

Boneh and Franklin suggest the use of a distributed PKG instead a single PKG to mitigate this problem. Their distributed PKG uses t out of m Shamir secret sharing [Sha79], which involves distributing the master key information among m PKGs, such that any $t+1$ of them, but no fewer, can compute the master key or generate a private key for a client. Instead of this basic arrangement, we suggest use of a verifiable and proactively secure distributed PKG over the Internet [KG07], where a master key is generated in a completely distributed way with each of m PKGs contributing a random share. The distributed design is additionally more robust; at any given time only $t+1$ of the m PKGs must be online in order for a client to retrieve his private key.

3.5 Applications of Our Anonymity Schemes

Our anonymous key agreement schemes can be used to perform anonymous communication in any setting having a BF-IBE setup. In recent years, numerous BF-IBE based solutions have been suggested for various practical situations such as ad-hoc networks [CL06, KKA03, SK05]. Our anonymous key agreement schemes can be used in all of these setups without any extra effort. As an example, we refer readers to the secure anonymous communication scheme for delay tolerant networks [KZH07]. In the present article, we focus on a new pairing-based onion routing protocol which achieves forward secrecy and constructs circuits without telescoping. We describe this protocol in the next section.

4 Pairing-Based Onion Routing

Low-latency onion routing requires one-way anonymous key agreement and forward secrecy. In this section, we describe a new pairing-based onion routing protocol using the non-interactive key agreement scheme defined in Section 3.2.

Our onion routing protocol has a significant advantage over the original onion routing protocol [GRS96] as well as the protocol used in Tor [DMS04]; it provides a practical way to achieve forward secrecy without building circuits by telescoping. Though this is possible with the original onion routing protocol, that method involves regularly communicating authenticated copies of ORs' public keys to the system users; forward secrecy is achieved by periodically rotating these keys. This does not scale well; every time the public keys are changed *all* users must contact a directory server to retrieve the new authenticated keys. However, our onion routing protocol uses ORs' identities, which users can obtain or derive without repeatedly contacting a central server, thus providing practical forward secrecy without telescoping.

4.1 Design Goals and Threat Model

As our protocol only differs from existing onion routing protocols in the circuit construction phase, our threat model is that of Tor. For example, adversaries have complete control over some part (but not all) of the network, as well as control over some of the nodes themselves.

We aim at frustrating attackers from linking multiple communications to or from a single user. Like Tor, we do not try to develop a system secure against a global observer, which can in theory follow end-to-end traffic. Further, it should not be feasible for any node to determine the identity of any node in a circuit other than its two adjacent nodes. Finally, we require forward secrecy: after some amount of time, the session keys used to protect node identities and the contents of messages are irrecoverable, even if all participants in the network are subsequently compromised.

4.2 Pairing-Based Onion Routing Protocol

An onion routing protocol involves a service provider, a set of onion routers, and users. In our protocol, a user does not build the circuit incrementally via telescoping, but rather in a single pass. The user chooses ℓ ORs from the available pool and generates separate pseudonyms for communicating with each of them. The user computes the corresponding session keys and uses them to construct a message with ℓ nested layers of encryption. This process uses the protocol given in Section 3.2 ℓ times.

The service provider works as the PKG for the ORs and provides private keys for their identities.

4.2.1 Forward Secrecy

There are two time-scale parameters in our protocol: the *master key validity period* (MKVP) and the *private key validity period* (PKVP). Both of these values relate to the forward secrecy of the system. The PKVP specifies how much exposure time a circuit has against compromises of the ORs that use it. That is, until the PKVP elapses, the ORs have enough information to collectively decrypt circuit construction onions sent during that PKVP. After each PKVP, ORs discard their current private keys and obtain new keys from the PKGs. This period can be short, perhaps on the order of an hour.

The MKVP specifies the circuit's exposure time against compromises of the (distributed) PKG which reveal the master secret s. Because changing s involves the participation of all of the ORs as well as the PKGs, we suggest the MKVP be somewhat longer than the PKVP, perhaps on the order of a day. Remember that in the t of m distributed PKG, if at least $m-t$ PKG members are honest and not compromised, no one will ever learn the value of a master secret.

4.2.2 Protocol Description

As discussed above, we propose the use of a distributed PKG, but for simplicity, our discussion will consider the PKG to be a single entity. Using a distributed PKG affects only the setup and key generation steps.

Setup Given the security requirements, the PKG generates a digital signature key pair (for any secure digital signature scheme). It also generates a prime n, two groups $\mathbb{G}$ (written additively) and $\mathbb{G}_T$ (written multiplicatively) of order n and a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Finally, the PKG chooses a full-domain cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}^*$. The PKG publishes all of these values except its private signature key.

Key Generation For each MKVP, the PKG generates a random master key $s \in \mathbb{Z}_n^*$ and a random $U \in \mathbb{G}$, and calculates sU. The PKG publishes a signed copy of (v_m, U, sU), where v_m is a timestamp for the MKVP in question. This U is a common value to be shared by all users of the system.

For every valid OR with identity $\texttt{OR}_i$, and for every PKVP v that overlaps with the MKVP, the PKG generates the private key $d_{vi} = sH(v||\texttt{OR}_i)$. The PKG distributes these private keys, as well as a copy of the signed (v_m, U, sU), to the appropriate ORs over a secure authenticated forward-secret channel. If an OR becomes compromised, the PKG can revoke it by simply no longer calculating its values of d_{vi}.

Note that this key distribution can be *batched*; that is, the PKG can precompute the master keys and private keys in advance (say a week at a time), and deliver them to the ORs in batches of any size from one PKVP at a time on up. This batching reduces the amount of time the PKG has to be online, and does not sacrifice forward secrecy. On the other hand, large batches will delay the time until a revocation becomes effective.

User Setup Once every MKVP v_m, each user must obtain a new signed tuple (v_m, U, sU) from any OR or from a public website. Once every PKVP v, the user computes the following pairing for each OR i and stores the results locally:

$$\gamma_{vi} = e(sU, Q_{vi}) = e(U, Q_{vi})^s \text{ where } Q_{vi} = H(v||\texttt{OR}_i)$$

Circuit Construction During a PKVP v, a user U chooses ℓ ORs (say $\texttt{OR}_1, \texttt{OR}_2, \ldots, \texttt{OR}_\ell$) and constructs a circuit $U \Leftrightarrow \texttt{OR}_1 \Leftrightarrow \texttt{OR}_2 \Leftrightarrow \cdots \Leftrightarrow \texttt{OR}_\ell$ with the following steps.

1. For each $\texttt{OR}_i$ in the circuit, the user generates a random integer $r_i \in \mathbb{Z}_n^*$ and computes the pseudonym $P_{Ui} = r_iU$ and the value ${\gamma_{vi}}^{r_i} = e(U, Q_{vi})^{sr_i}$. From ${\gamma_{vi}}^{r_i}$ two session

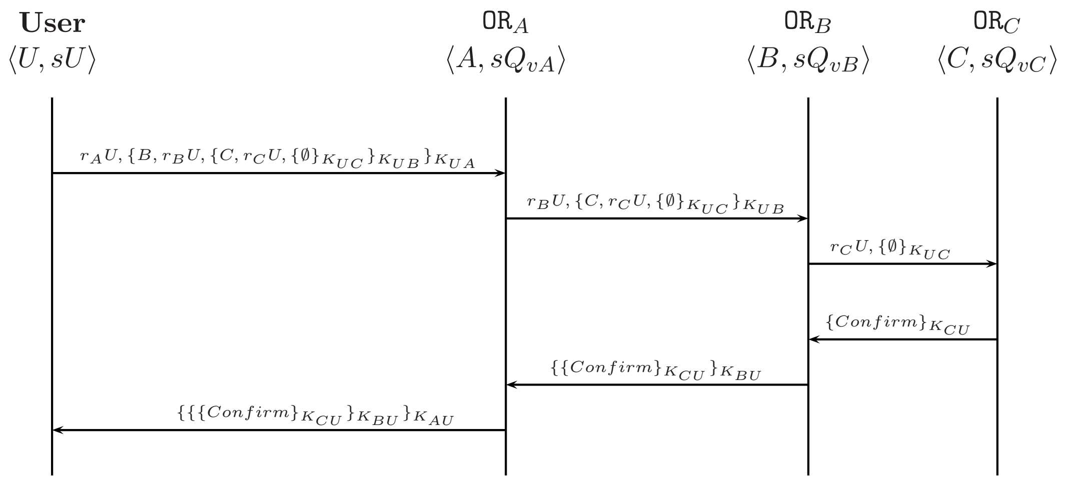

User
$\langle U, sU\rangle$
$\mathtt{OR}_A$
$\langle A, sQ_{vA}\rangle$
$\mathtt{OR}_B$
$\langle B, sQ_{vB}\rangle$
$\mathtt{OR}_C$
$\langle C, sQ_{vC}\rangle$
$r_AU, \{B, r_BU, \{C, r_CU, \{\emptyset\}_{K_{UC}}\}_{K_{UB}}\}_{K_{UA}}$
$r_BU, \{C, r_CU, \{\emptyset\}_{K_{UC}}\}_{K_{UB}}$
$r_CU, \{\emptyset\}_{K_{UC}}$
$\{Confirm\}_{K_{CU}}$
$\{\{Confirm\}_{K_{CU}}\}_{K_{BU}}$
$\{\{\{Confirm\}_{K_{CU}}\}_{K_{BU}}\}_{K_{AU}}$

Figure 1: A user builds a circuit with three ORs.

keys are derived: a forward session key K_{Ui} and a backward session key K_{iU}. Finally, the following onion is built and sent to $\mathtt{OR}_1$, the first OR in the circuit:

$$r_1U, \{\mathtt{OR}_2, r_2U, \{\cdots\{\mathtt{OR}_\ell, r_\ell U, \{\emptyset\}_{K_{U\ell}}\}\cdots\}_{K_{U2}}\}_{K_{U1}}$$

Here $\{\cdots\}_{K_{Ui}}$ is symmetric-key encryption and $\emptyset$ is an empty message which informs $\mathtt{OR}_\ell$ that it is the exit node.

2. After receiving the onion, the OR with identity $\mathtt{OR}_i$ uses the received r_iU and its currently valid private key d_{vi} to compute $e(r_iU, d_{vi}) = e(U, Q_i)^{r_i s} = {\gamma_{vi}}^{r_i}$. It derives the forward session key K_{Ui} and the backward session key K_{iU}. It decrypts the outermost onion layer $\{\cdots\}_{K_{Ui}}$ to obtain the user's next pseudonym, the nested ciphertext, and the identity of the next node in the circuit. The OR then forwards the pseudonym and ciphertext to the next node. To avoid replay attacks, it also stores pseudonyms (see Section 7). The process ends when an OR ($\mathtt{OR}_\ell$ in this case) gets $\emptyset$.
3. The exit node $\mathtt{OR}_\ell$ sends a confirmation message encrypted with the backward session key $\{Confirm\}_{K_{\ell U}}$ to the previous OR in the circuit. Each OR encrypts the confirmation with its backward session key and sends it to the previous node, until the ciphertext reaches the user. The user decrypts the ciphertext layers to verify the confirmation.
4. If the user does not receive the confirmation in a specified time, she selects a different set of ORs and repeats the protocol.

The circuit construction is further illustrated in Figure 1, where a user builds a three-node circuit.

Anonymous Communication After the circuit is constructed, communication proceeds in the same manner as Tor. The user sends onions through the circuit with each layer encrypted with the forward keys K_{Ui}, and each hop decrypts one layer. Replies are encrypted at each hop with the backward key K_{iU}, and the user decrypts the received onion.

Note that as an optimization, one or more messages can be bundled inside the original circuit construction onion, in place of $\emptyset$.

We analyze the security for the above protocol in Section 6.

4.3 Advantages Over First-Generation Onion Routing

As discussed earlier, it is possible to achieve forward secrecy in first-generation onion routing by periodically replacing the public-private key pairs of the ORs. Following the change, the service provider publishes signed copies of the new OR public keys after getting authentic copies from the ORs. However, this requires all users to regularly obtain fresh authenticated public key information for all ORs.

In contrast, with our system, each user only needs to obtain the single authenticated value (v_m, U, sU), and only once every MKVP. The user can then calculate the required γ_{vi} values on her own until the end of that period, thus reducing the load on the service provider. This load is further reduced by having the service provider never communicate directly with users at all, but only with the ORs.

As a consequence, our pairing-based onion routing is a more practical solution for low-latency anonymous communication.

4.4 Advantages Over Telescoping in Tor

The Tor network, in practice, uses the telescoping approach based on the Diffie-Hellman key exchange to form an anonymity circuit. We find the following advantages for our protocol over the telescoping approach.

- Although our above-defined protocol requires occasional private key generation for ORs to achieve forward secrecy, it saves communication cost at every circuit construction by avoiding telescoping. We discuss our communication and computational advantages in Section 8.5.

- The absence of telescoping in our protocol provides flexibility to the user to modify a circuit on the fly. For example, suppose a user U has constructed a circuit $(U \Leftrightarrow \mathtt{OR}_1 \Leftrightarrow \mathtt{OR}_2 \Leftrightarrow \cdots \Leftrightarrow \mathtt{OR}_i \Leftrightarrow \cdots \Leftrightarrow \mathtt{OR}_\ell)$. In our protocol, she can bundle instructions to immediately replace $\mathtt{OR}_i$ with $\mathtt{OR}'_i$ in the next message, while keeping the remaining circuit intact. Her circuit would then be $(U \Leftrightarrow \mathtt{OR}_1 \Leftrightarrow \mathtt{OR}_2 \Leftrightarrow \cdots \Leftrightarrow \mathtt{OR}'_i \Leftrightarrow \cdots \Leftrightarrow \mathtt{OR}_\ell)$.

4.5 Issues with the Proposed Scheme

The certifying authorities in the Tor system need to be less trusted than the PKG in our scheme. It is also possible for $t+1$ malicious PKGs to passively listen to all of the traffic as they can compute private keys for all ORs. A geographically and politically distributed implementation of m PKGs certainly reduces this possibility.

To passively decrypt an OR's messages, an adversary of the Tor system must know the OR's private key, as well as the current Diffie-Hellman key (established for each circuit). In our scheme, as it is non-interactive, an adversary who knows only the OR's private key can decrypt all of the messages for that OR. This may be an acceptable trade-off, considering the advantages gained from the non-interactive protocol.

Further, this onion routing circuit construction provides forward secrecy, only after ORs' private keys are rotated. In other words (as defined by Øverlier and Syverson [ØS07]), it only provides *eventual* forward secrecy rather than *immediate* forward secrecy. Consequently, it has shorter PKVP as compared to the key replacement period in Tor and PKGs (any $t+1$ of them) need to be online with greater reliability. If fewer than $t+1$ PKGs are available, the whole system is paralysed after the current batch. In the next section, we resolve this issue, without a significant increase in circuit construction time, by introducing a partially interactive onion routing circuit construction.

5 λ-pass Onion Routing

Tor achieves immediate forward secrecy using telescoping. Telescoping can also be considered as an ℓ-pass circuit construction, where ℓ is the circuit length, with immediate forward secrecy at each of the OR nodes. In practice however, it is sufficient to have immediate forward secrecy at fewer than ℓ nodes, as an adversary will be stymied when it encounters any such node. In this section, we define λ-pass onion routing circuit construction, which achieves immediate forward secrecy at λ nodes (for $2 < \lambda \leq \ell$) with reduced circuit construction cost over telescoping.

5.1 Impossibility of Immediate Forward Secrecy in Single-pass Circuit Construction

To motivate multiple pass circuit construction, we will describe why it is impossible to obtain immediate forward secrecy in any single-pass circuit construction, regardless of the cryptographic setting.

In an immediately forward secret circuit construction, compromise of OR private keys after a circuit is built should not allow any information about the circuit path to be recovered. Further, after the circuit is destroyed and the keys are dropped, it should not be possible for any honest user and an OR to re-compute their shared keys for that session. To achieve these properties, *both* parties must contribute some randomness to the creation of the the session key and they must drop these random values once the session keys are generated. Consequently, before the user can generate the forward session key, the random values (in some modified form) have to be exchanged between the user and the OR. The modified forms should enable the only the authentic receiver to compute the session key. In an immediate forward secret circuit construction like Tor, these session-dependent random values are realized using the Diffie-Hellman exponents (x, y), while the Diffie-Hellman parameters (g^x, g^y) provide the publicly exchanged forms of the randomness.

In any single-pass circuit construction, an OR does not reply immediately after receiving an onion (except for exit nodes). Therefore, addition of any randomness from the OR in the forward session key is not possible, before that session key can be used to convey the OR its successor. Consequently, any time later in the same PKVP, an adversary can compromise the OR, use the OR's private key to generate the session keys and exploit those to find the next node in the circuit path. Further, although it is possible for nodes to send their part of randomness for session keys along with the (backward) confirmation onion, this does not provide any advantage as the adversary can always find the circuit path by decrypting the forward onion. Thus we see that it is not possible to obtain immediate forward secrecy in a single-pass circuit construction.

5.2 λ-pass Circuit Construction

As replies from the last node of single-pass circuit constructions are direct, immediate forward secrecy is easy to achieve at this node. Øverlier and Syverson [ØS07] observe that there is always forward secrecy at the entry node, as the link between the user and the entry node in the circuit is encrypted using TLS. Therefore, our circuit construction defined in Section 4 can easily be a 2-pass circuit construction with immediate forward secrecy at the entry node and the exit node. Here, we consider $\lambda - 2$ additional single-pass circuit constructions to achieve immediate forward secrecy at λ nodes. We note that Øverlier and Syverson [ØS07, Protocol 3] propose a similar circuit construction, but their focus is on dealing with replay attacks. Our λ-pass protocol adds message flows to provide partial immediate forward secrecy, whereas their protocol uses an increased number of flows to prevent replay of construction onions.

5.2.1 Protocol Description

Although our λ-pass circuit construction can be applied in any public-key setting, for simplicity, here we present it for pairing-based onion routing, as defined in Section 4.

Setup, Key Generation, User Setup These are as described in Section 4.2.2.

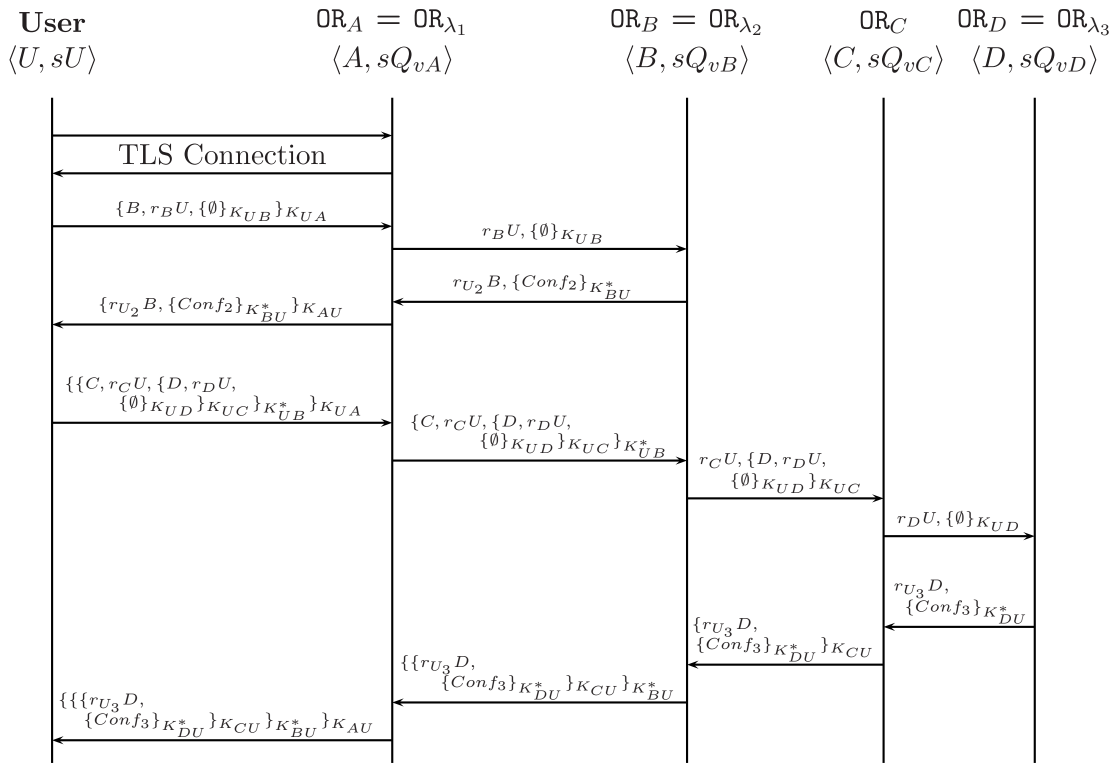
Circuit Construction During a PKVP v, a user U chooses a set of ORs (say $\mathtt{OR}_1, \mathtt{OR}_2, \ldots, \mathtt{OR}_\ell$) and constructs a circuit $U \Leftrightarrow \mathtt{OR}_1 \Leftrightarrow \mathtt{OR}_2 \Leftrightarrow \cdots \Leftrightarrow \mathtt{OR}_\ell$ with the following steps.

1. The user selects λ indices $1 = \Lambda_1 < \Lambda_2 < \cdots < \Lambda_{\lambda-1} < \Lambda_\lambda = \ell$.
2. As in Section 4.2.2, for each $\mathtt{OR}_i$ in the circuit, the user generates a random integer $r_i \in \mathbb{Z}_n^*$ and computes the pseudonym $P_{Ui} = r_i U$ and the value ${\gamma_{vi}}^{r_i} = e(U, Q_{vi})^{sr_i}$. From ${\gamma_{vi}}^{r_i}$ two session keys are derived: a forward session key K_{Ui} and a backward session key K_{iU}. At this stage, the user forgets the random values r_i for all but $r_{\Lambda_1}, r_{\Lambda_2}, \ldots, r_{\Lambda_\lambda}$.
3. The user forms a TLS connection with $\mathtt{OR}_{\Lambda_1} = \mathtt{OR}_1$, which generates the session keys K_{U1} and K_{1U}. This TLS connection must use one of the ephemeral Diffie-Hellman ciphersuites guaranteeing forward secrecy for this first connection [DR06].
4. The user then creates the following onion and sends it to $\mathtt{OR}_1$.

$$\{\mathtt{OR}_2, r_2U, \{\cdots\{\mathtt{OR}_{\Lambda_2}, r_{\Lambda_2}U, \{\emptyset\}_{K_{U\Lambda_2}}\}\cdots\}_{K_{U2}}\}_{K_{U1}}$$

   Here $\{\cdots\}_{K_{Ui}}$ is symmetric-key encryption and $\emptyset$ is an empty message which informs $\mathtt{OR}_{\Lambda_1}$ that it is the exit node.
5. $\mathtt{OR}_1$ uses K_{U1} and decrypts the onion to determine the next node $\mathtt{OR}_2$. The later nodes $\mathtt{OR}_i$ with $i \geq 2$, use r_iU and their currently valid private keys d_{vi} to compute $e(r_iU, d_{vi}) = e(U, Q_i)^{r_is} = {\gamma_{vi}}^{r_i}$. They derive the forward session keys K_{Ui} and the backward session keys K_{iU}. They decrypt the outermost onion layer $\{\cdots\}_{K_{Ui}}$ to obtain the user's next pseudonym, the nested ciphertext, and the identity of the next node in the circuit. The ORs then forward the pseudonym and ciphertext to the next node. To avoid replay attacks, they also store pseudonyms (see Section 7). The process ends when $\mathtt{OR}_{\Lambda_2}$ gets $\emptyset$.
6. The last node in the partial circuit $\mathtt{OR}_{\Lambda_2}$ generates a random integer $r_{U_2} \in \mathbb{Z}_n^*$, and computes a pseudonym $r_{U_2}Q_{v\Lambda_2}$. It then generates ${\gamma_{v\Lambda_2}^{r_{\Lambda_2}}}^{r_{U_2}}$ and derives modified forward and backward session keys ($K^*_{U\Lambda_2}$ and $K^*_{\Lambda_2U}$) and sends a confirmation message encrypted with the backward session key $\{Confirm\}_{K_{\Lambda_2U}}$ along with the pseudonym $r_{U_2}Q_{v\Lambda_2}$ to the previous OR in the circuit. To obtain immediate forward secrecy, it also forgets the random integer r_{U_2} and the value ${\gamma_{v\Lambda_2}^{r_{\Lambda_2}}}^{r_{U_2}}$ right away, and will forget $K^*_{U\Lambda_2}$ and $K^*_{\Lambda_2U}$ immediately after the circuit is no longer in use.
7. Each OR encrypts the confirmation with its backward session key and sends it to the previous node, until the ciphertext reaches the user. The user decrypts the ciphertext layers to verify the confirmation and in the process, generates the modified session keys for $\mathtt{OR}_{\Lambda_2}$ using the received pseudonym $r_{U_2}Q_{v\Lambda_2}$ and the stored random value r_{Λ_2}. After this, the user drops the random value r_{Λ_2}.
8. Now, the partial circuit $U \Leftrightarrow A \cdots \Leftrightarrow \mathtt{OR}_{\Lambda_2}$ is used to extend the circuit to $\mathtt{OR}_{\Lambda_3}$ by sending the following onion to $\mathtt{OR}_1$.

$$\{\{\cdots\{\mathtt{OR}_{\Lambda_2+1}, r_{\Lambda_2+1}U, \{\cdots\{\mathtt{OR}_{\Lambda_3}, r_{\Lambda_3}U, \{\emptyset\}_{K_{U\Lambda_3}}\}\cdots\}_{K_{U\Lambda_2+1}}\}_{K_{U\Lambda_2}}\cdots\}_{K_{U2}}\}_{K_{U1}}$$



User
$\langle U, sU\rangle$
$\mathtt{OR}_A = \mathtt{OR}_{\lambda_1}$
$\langle A, sQ_{vA}\rangle$
$\mathtt{OR}_B = \mathtt{OR}_{\lambda_2}$
$\langle B, sQ_{vB}\rangle$
$\mathtt{OR}_C$
$\langle C, sQ_{vC}\rangle$
$\mathtt{OR}_D = \mathtt{OR}_{\lambda_3}$
$\langle D, sQ_{vD}\rangle$
TLS Connection
$\{B, r_BU, \{\emptyset\}_{K_{UB}}\}_{K_{UA}}$
$r_BU, \{\emptyset\}_{K_{UB}}$
$r_{U_2}B, \{Conf_2\}_{K^*_{BU}}$
$\{r_{U_2}B, \{Conf_2\}_{K^*_{BU}}\}_{K_{AU}}$
$\{\{C, r_CU, \{D, r_DU, \{\emptyset\}_{K_{UD}}\}_{K_{UC}}\}_{K^*_{UB}}\}_{K_{UA}}$
$\{C, r_CU, \{D, r_DU, \{\emptyset\}_{K_{UD}}\}_{K_{UC}}\}_{K^*_{UB}}$
$r_CU, \{D, r_DU, \{\emptyset\}_{K_{UD}}\}_{K_{UC}}$
$r_DU, \{\emptyset\}_{K_{UD}}$
$r_{U_3}D, \{Conf_3\}_{K^*_{DU}}$
$\{r_{U_3}D, \{Conf_3\}_{K^*_{DU}}\}_{K_{CU}}$
$\{\{r_{U_3}D, \{Conf_3\}_{K^*_{DU}}\}_{K_{CU}}\}_{K^*_{BU}}$
$\{\{\{r_{U_3}D, \{Conf_3\}_{K^*_{DU}}\}_{K_{CU}}\}_{K^*_{BU}}\}_{K_{AU}}$

Figure 2: A user builds a circuit with four ORs and $\lambda = 3$.

9. The user completes λ passes to construct the complete circuit $U \Leftrightarrow \mathtt{OR}_1 \Leftrightarrow \mathtt{OR}_2 \Leftrightarrow \cdots \Leftrightarrow \mathtt{OR}_\ell$.
10. If the user does not receive any of the λ confirmations in specified times, she selects a different set of ORs and repeats the protocol.

Anonymous Communication is as described in Section 4.2.2.

This circuit construction is further illustrated in Figure 2, where a user builds a four-node circuit with $\lambda = 3$.

We observe that the security and anonymity properties of this protocol are straight-forward extensions to those of our single-pass construction in Section 4; we elaborate on security in Section 6.

5.2.2 Value of λ and node-placement

For $\lambda = \ell$, the above circuit becomes a telescoping circuit construction (like Tor) and for $\lambda = 2$, with minor modifications, it is essentially the single-pass construction of Section 4. Considering an adversary who controls some of the OR network (but not all, as in our threat model, Section 4.1), it is certainly advantageous to keep $2 < \lambda < \ell$. We observe that assuming only cryptographic attacks, it is sufficient to have $\lambda = 3$; that is, immediate forward secrecy at the entry node, exit node and one of the nodes in between. In this case, the adversary's successive compromise of the ORs in a circuit is thwarted once it reaches the immediate forward secrecy node. In other words, assuming that the adversary can compromise all ORs in the network, it cannot link the two parts of the circuit divided at the immediate forward secret node. However, with traffic analysis and correlation attacks, this might not be sufficient as the attacker could access all traffic to and from an immediate forward secrecy node. In practice, choosing the value of λ requires an estimate of how many nodes the adversary can compromise, as well as how much traffic he can observe.

An immediate question is the placement of the immediate forward secrecy nodes in the circuit path. It is easy to observe that a circuit with two adjacent immediate forward secrecy nodes is more difficult to attack using traffic analysis than one where those two nodes are separated. Further, as the ultimate goal of onion routing is anonymity for the sender and receiver, it is good to have immediate forward secrecy nodes at the start and at the end of the circuit. Therefore, we suggest $\lceil \lambda/2 \rceil$ immediate forward secrecy nodes at the start of the circuit and remaining $\lfloor \lambda/2 \rfloor$ immediate forward secrecy nodes at the end of the circuit. In cases when the recipient does not require anonymity (e.g. it is a webserver), the efficiency of the construction can be improved by placing the first $\lambda - 1$ nodes closest to the sender; that is, by selecting $\Lambda_i = i$ for $1 \leq i \leq \lambda - 1$ and $\Lambda_\lambda = \ell$. An attacker who observes the onion past the last forward secrecy node may be able to decrypt the remaining layers, but the first $\lambda - 1$ forward secrecy nodes have already provided anonymity for the sender.

Once the random parameters r_{U_i} and r_{Λ_i} are dropped by the $\texttt{OR}_{\Lambda_i}$ and the user U respectively, deriving their session keys becomes the BDH problem, even if $\texttt{OR}_{\Lambda_i}$ gets compromised during the PKVP. Therefore, we achieve immediate forward secrecy at λ ORs and the PKVP could be longer, or made equal to the MKVP. The latter would also eliminate the need to attach validity periods to OR identities.

6 Security Analysis

Camenisch and Lysyanskaya [CL05] give a protocol for onion routing with provable security in the universal composability (UC) model [Can01]. The UC model imposes additional overhead on their protocol. We aim for a simpler and more efficient protocol at the expense of provability in the UC model. Nevertheless, we prove some security properties of our protocol. Here is a list of the properties we consider; detailed definitions will follow.

Cryptographic unlinkability This property ensures that a circuit, which has at least one honest node, provides unlinkability between a sender and a receiver. This property allows us to use a strong attack model; anonymity is still possible when the adversary controls all but one of the routers in the path. By the term *cryptographic unlinkability* we exclude network-level linking attacks.

Integrity and correctness These properties are defined by Camenisch and Lysyanskaya [CL05]. An onion routing scheme has the *correctness* property if a message reaches the intended recipient whenever the an onion is i) formed correctly, ii) processed by the right routers in the right order, and iii) these routers follow the protocol. Integrity is achieved if onions longer than some upper limit on the length can be recognized by routers. We observe that our circuit construction trivially achieves these correctness and integrity properties.

Key secrecy An attacker controlling all but one honest node in a circuit should not be able to recover the secret key shared between the user and the honest node. Since this is effectively a run of the protocol in Section 3.2 between the user and the honest node, the security proof in Section 3.3.2 applies here as well.

Circuit position secrecy Other works on the subject [CL05, Möl03] desire the following property, which we term *circuit position secrecy.* When a router receives an onion, it should not be possible for it to learn which position it has in a circuit (unless it is the entry or exit node). We discuss some ways to provide circuit position secrecy.

Note that when proving security using the above properties, our λ-pass circuit construction in Section 5.2, can be considered equivalent to λ passes of our single-pass circuit construction protocol. Therefore, we do not consider it seperately in our security discussion.

6.1 Cryptographic Unlinkability

Suppose an adversary controls all routers in a circuit of length ℓ except for one honest router, H. By collapsing the adversarial routers into a single entity, we can distill this scenario into:

$$\text{User } U \rightarrow A_1 \rightarrow H \rightarrow A_2 \rightarrow \text{Recipient R}$$

where nodes A_1 and A_2 are controlled by the adversary. When U sends an onion to A_1, it is processed and forwarded to H who processes it and forwards it to A_2. If A_2 can determine that the onion received from H was also processed by A_1, then U and R can be linked. Traffic analysis and timing/correlation attacks may be able to link U and R, but in this section we show that there is no *cryptographic* linking possible.

When there are multiple users in the system (a basic assumption for all anonymity systems), the problem can be illustrated as follows.

$$\begin{tikzcd}
U \arrow[r] & A_1 \arrow[dr] & & A_2 \arrow[r] & R \\
& & H \arrow[ur] \arrow[dr] & & \\
U' \arrow[r] & A'_1 \arrow[ur] & & A'_2 \arrow[r] & R'
\end{tikzcd}$$

We will show that this problem is equivalent to distinguishing ciphertexts.

The situation depicted above is captured by the following security game, which we call the *cryptographic unlinkability game*.

Setup The challenger $\mathcal{C}$ performs the setup described by the onion routing protocol. The adversary $\mathcal{A}$ chooses $m, \mathtt{ID}_{A_1}, \mathtt{ID}_{A'_1}, \mathtt{ID}_{A_2}, \mathtt{ID}_{A'_2}, \mathtt{ID}_H$. $\mathcal{C}$ sends the private keys for $\mathtt{ID}_{A_1}, \mathtt{ID}_{A'_1}, \mathtt{ID}_{A_2}, \mathtt{ID}_{A'_2}$, to $\mathcal{A}$. It is assumed that router identities are of a fixed length to avoid attacks based on the ciphertext sizes.

Create onions The challenger chooses $b \in \{0, 1\}$ at random.
If $b = 0$, $\mathcal{C}$ sends to $\mathcal{A}$:

$$\begin{aligned}
O_1 &\text{ for circuit } U \rightarrow A_1 \rightarrow H \rightarrow A_2 \\
&= \mathtt{ID}_{A_1}, r_{A_1}U, \{\mathtt{ID}_H, r_H U, \{\mathtt{ID}_{A_2}, r_{A_2}U, \{m\}_{K_{UA_2}}\}_{K_{UH}}\}_{K_{UA_1}} \\
O'_1 &\text{ for circuit } U \rightarrow A'_1 \rightarrow H \rightarrow A'_2 \\
&= \mathtt{ID}_{A'_1}, r_{A'_1}U, \{\mathtt{ID}_H, r'_H U, \{\mathtt{ID}_{A'_2}, r_{A'_2}U, \{m\}_{K_{UA'_2}}\}_{K_{UH'}}\}_{K_{UA'_1}}
\end{aligned}$$

and if $b = 1$, $\mathcal{C}$ sends to $\mathcal{A}$:

$$\begin{aligned}
O_1 &\text{ for circuit } U \rightarrow A_1 \rightarrow H \rightarrow A'_2 \\
&= \mathtt{ID}_{A_1}, r_{A_1}U, \{\mathtt{ID}_H, r_H U, \{\mathtt{ID}_{A'_2}, r_{A'_2}U, \{m\}_{K_{UA'_2}}\}_{K_{UH}}\}_{K_{UA_1}} \\
O'_1 &\text{ for circuit } U \rightarrow A'_1 \rightarrow H \rightarrow A_2 \\
&= \mathtt{ID}_{A'_1}, r_{A'_1}U, \{\mathtt{ID}_H, r'_H U, \{\mathtt{ID}_{A_2}, r_{A_2}U, \{m\}_{K_{UA_2}}\}_{K_{UH'}}\}_{K_{UA'_1}} \ .
\end{aligned}$$

A_1 **processing** The node A_1 processes the onions O_1 and O'_1 before passing them along to H (A_1 may or may not follow the protocol). The resulting onions O_H and O'_H respectively are then returned to $\mathcal{C}$.

H **processing** $\mathcal{C}$ removes the layer encrypted for H. If the decryptions are invalid the game is aborted ($\mathcal{C}$ will know since it created the onions). If the decryptions are valid, $\mathcal{C}$ returns the resulting O_2, O'_2 to $\mathcal{A}$. Here, O_2 and O'_2 are the output onions destined for A_2 and A'_2 respectively.

Guess $\mathcal{A}$ must determine which input onion corresponds to which output onion by outputting a guess bit b'. We say that the adversary $\mathcal{A}$ (t, ϵ)-wins the game if $\mathcal{A}$ outputs b' in time t such that

$$\text{Prob}(b = b') = 1/2 + \epsilon \ .$$

Winning this cryptographic unlinkability game can be reduced to distinguishing ciphertexts. To achieve that, we first present the **IND-CPA** game for (indistinguishability in a chosen plaintext attack).

Setup Let E_K be a symmetric encryption function which uses key K. $\mathcal{C}$ chooses K at random from the keyspace associated with E.

Extraction Queries An adversary $\mathcal{A}_{\text{CPA}}$ is given oracle access to E_K. She can obtain ciphertexts for polynomially bounded number of plaintext messages of her choice any time during the game.

Challenge The $\mathcal{A}_{\text{CPA}}$ chooses two equal length plaintexts m_0, m_1, and sends them to $\mathcal{C}$. $\mathcal{C}$ chooses $b \in \{0, 1\}$ at random, and sends $E_K(m_b)$ to $\mathcal{A}_{\text{CPA}}$.

Guess $\mathcal{A}_{\text{CPA}}$ outputs a guess $b' \in \{0, 1\}$. We say that $\mathcal{A}_{\text{CPA}}$ (t, ϵ)-wins the **IND-CPA** game if $\mathcal{A}_{\text{CPA}}$ returns b' in time t such that

$$\text{Prob}(b = b') = 1/2 + \epsilon \ .$$

A similar game, which we call the **simultaneous-IND-CPA** game, arises from the cryptographic unlinkability problem described above.

Setup Let E_K be an encryption function which uses key K. $\mathcal{C}$ chooses K_1, K_2 at random from the keyspace associated with E.

Extraction Queries An adversary $\mathcal{A}_{\text{Simul-CPA}}$ is given oracle access to E_{K_1} and E_{K_2}. She obtains ciphertexts for polynomially bounded number of plaintext messages of her choice any time during the game.

Challenge $\mathcal{A}_{\text{Simul-CPA}}$ chooses equal length plaintexts m_0, m_1, and sends them to $\mathcal{C}$. $\mathcal{C}$ chooses $b \in \{0, 1\}$ at random, and sends $E_{K_1}(m_b)$ and $E_{K_2}(m_{\overline{b}})$ to $\mathcal{A}_{\text{Simul-CPA}}$.

Guess $\mathcal{A}_{\text{Simul-CPA}}$ outputs a guess $b' \in \{0, 1\}$. We say that $\mathcal{A}_{\text{Simul-CPA}}$ (t, ϵ)-wins the **simultaneous-IND-CPA** game if $\mathcal{A}_{\text{Simul-CPA}}$ returns b' in time t such that

$$\text{Prob}(b = b') = 1/2 + \epsilon \ .$$

We say an encryption function E is (**simultaneous-**)**IND-CPA** if for any adversary $\mathcal{A}$ which (t, ϵ)-wins the (**simultaneous-**)**IND-CPA** game, ϵ is negligible if t is polynomial.

Lemma 6.1. *If E is a family of pseudorandom permutations, then E is* **IND-CPA** *if and only if it is* **simultaneous-IND-CPA***.*

Proof. First assume an adversary $\mathcal{A}_{\text{CPA}}$, which can (t, ϵ)-win an **IND-CPA** game $(m_0, m_1, E_K(m_b))$. Given an access to such an adversary, an **simultaneous-IND-CPA** game $(m_0, m_1, E_{K_1}(m_b), E_{K_2}(m_{\bar{b}}))$ can be (t, ϵ)-won simply by sending the query $(m_0, m_1, E_{K_1}(m_b))$ to the adversary $\mathcal{A}_{\text{CPA}}$. Therefore, E is **simultaneous-IND-CPA** $\Rightarrow$ E is **IND-CPA**.

Now assume E is a family of pseudorandom permutations and is **IND-CPA**, and suppose there is an adversary $\mathcal{A}_{\text{Simul-CPA}}$, which (t, γ)-wins the **simultaneous-IND-CPA** game $(m_0, m_1, E_{K_1}(m_b), E_{K_2}(m_{\bar{b}}))$ for non-negligible γ. For an **IND-CPA** input $(m_0, m_1, E_K(m_i))$, we present $\mathcal{A}_{\text{Simul-CPA}}$ with the **simultaneous-IND-CPA** instance $(m_0, m_1, E_K(m_i), \$)$ where $\$$ is a random string of length $|E_K(m_i)|$, and with oracle access to E_{K_2} for a randomly chosen key K_2. Suppose $\mathcal{A}_{\text{Simul-CPA}}$ returns i with probability $1/2 + \kappa$. It follows that $\mathcal{A}_{\text{Simul-CPA}}$ can be used to distinguish encryptions of E with key K_2 from randomly chosen strings with probability $1/2 + \mu$, where $\mu = |\gamma - \kappa|$. Since E is assumed to be a psuedorandom permutation, μ must be negligible, and $\kappa \geq \gamma - \mu$. Therefore, E is **IND-CPA** $\Rightarrow$ E is **simultaneous-IND-CPA**. □

Theorem 6.2. *If E is a family of pseudorandom permutations which is* **IND-CPA***, then for any adversary which (t, ϵ)-wins the cryptographic unlinkability game, if t is polynomial, then ϵ is negligible.*

Proof. Suppose $\mathcal{A}$ is playing the unlinkability game; we focus on the Guess step. $\mathcal{A}$ has O_H and O'_H (the onions processed by $\mathcal{C}$), which are not encryptions of m (which he chose), but encryptions $M_1 = \{m\}_{K_{UA_1}}$ and $M_2 = \{m\}_{K_{UA_2}}$ known to $\mathcal{A}$. This gives us an instance of **simultaneous-IND-CPA** except $\mathcal{A}$ did not get to choose the messages. If $\mathcal{A}$ (t, ϵ)-solves this instance for non-negligible ϵ, $\mathcal{A}$ can certainly (t, ϵ)-solve instances where $\mathcal{A}$ chooses M_1, M_2. By Lemma 6.1 $\mathcal{A}$ solves an instance of **IND-CPA** as well. □

Therefore, if our encryption function is indistinguishable in a chosen plaintext attack, then our onion routing protocol has cryptographic unlinkability (in our model of multiple users and one honest router per circuit). Since key secrecy is proven for our construction, any symmetric key encryption which provides indistinguishability given key secrecy will give our protocol cryptographic unlinkability.

6.2 Circuit Position Secrecy

We now treat the issue of preventing routers from learning which position they have in a circuit, when an adversary controls all but one router. Note that an adversary who controls all but one router in a circuit, and who knows the circuit length is always able to deduce which position his routers have in *some* circuit. Adversarial nodes "sandwiched" between two honest nodes learn nothing about their position.

The technique used to keep the circuit position secret in the construction of Camenisch and Lysyanskaya [CL05] can be used with the protocol of Section 4. The method handles the fact that the onion gets smaller as it moves down the path, allowing routers to learn their position. Intuitively, the idea is to send a group of equal size onions, each one encrypting data destined for one of the nodes along the path. There is no intermediate data between layers of the onions, but only at their centres. In a three-node circuit, we would have a one-layer, two-layer and three-layer onion, and each router would remove a layer from each. After receiving its data, the router replaces the onion it was able to read with a random string, and forwards that string and the other two onions to the next router.

7 Systems Issues

In this section, we describe how components of an onion routing system such as Tor would behave in a pairing-based setting. To implement pairings, we must choose groups where pairings are known, and are efficiently computable. Once these groups are fixed we can estimate the computational cost required to construct a circuit. The next section will compare the costs of our schemes to the cost of setting up a circuit in Tor.

PKG

As discussed in Section 3.4, the PKG should be distributed across servers run by independent parties. To provide robustness, a "t of m" secret sharing scheme may be employed; this would mean that an OR need only contact $t + 1$ of m "pieces" of the PKG to learn its complete private key. Naturally, private key information must always be communicated over a secure channel. We note that end users of the system will have no reason to contact the PKG; the PKG only communicates with ORs, and sends one private key (an element of $\mathbb{G}$) per PKVP to each. The load on the PKG should therefore be quite manageable. For added protection from attack, the PKG could even situate itself as a "hidden service" [DMS04, Sec. 5], so that only known ORs could even connect to it, and no one would know where many of the pieces were located.

Channel Security

The security and forward secrecy depends on the channel between the PKG and the OR used to compute the private key. With a non-distributed PKG, an attacker can compromise an OR's private key by compromising this channel. The distributed PKG provides robustness here as well, since the attacker must subvert $t+1$ secure channels to reconstruct the private key from the shares.

Onion Router Identities

Users calculate γ_{vi} based on each router's identity $\texttt{ID}_i$. This identity can be as simple as a port number and a hostname or IP address. In that case, the BF-IBE setup ensures that if a user knows how to contact an OR, she automatically knows its public key.

The value γ_{vi} is also based on the current PKVP v. To avoid requiring tight synchronization between the clocks of ORs and users, ORs should keep their private keys d_{vi} around for a short time after the official end of the PKVP, but must securely discard them after that.

Replay Prevention

To avoid attacks where adversaries replay old circuit construction onions, ORs should store the pseudonyms they receive for the duration of a PKVP and drop onions which re-use a pseudonym. After circuit construction, replay attacks can be prevented with existing methods [DM08].

Directory Servers

Directory servers can be used to provide signed information about the list of available ORs to the users of the system. The directory servers in Tor, for example, provide a list of the ORs along with their public keys, status, capabilities and policies. In our pairing-based setting, of course, the public keys are unnecessary.

8 Performance

In this section, we consider the cost of single pass and λ-pass circuit constructions from a user through ℓ onion routers. We estimate the computational cost, and count the number of AES-encrypted network communications. We compare the performance of our systems to that of Tor.

8.1 Security Levels and Parameter Sizes

Before comparing the costs of the cryptography in the schemes we determine the parameter sizes required to provide the same level of security currently provided by Tor.

Tor uses public key parameters to provide security at the 80-bit level [Gol06]. The discrete log problem is in a 1024-bit field, and the RSA problem uses a 1024-bit modulus. The symmetric parameters provide significantly more security, by using AES with a 128-bit key.

We must choose appropriate groups $\mathbb{G}$ and $\mathbb{G}_T$ over which our pairing will be defined in order to offer similar strength. The current favourite choice is the group of torsion points of an elliptic curve group over a finite field, with either the Weil or Tate pairing. To achieve an 80-bit security level, the elliptic curve discrete log problem an attacker faces must be in a group of at least 160 bits. Due to the reduction of Menezes, Okamoto and Vanstone [MOV91], we must also ensure that discrete logs are intractable in the target group, $\mathbb{G}_T$. In our case, $\mathbb{G}_T = \mathbb{F}_{p^k}$, where k is the embedding degree of our curve taken over $\mathbb{F}_p$. We must then choose our curve E, a prime p, and embedding degree k such that $E(\mathbb{F}_p)$ has a cyclic subgroup of prime order $n \approx 2^{160}$, and p^k is around 2^{1024}. This can be achieved in a variety of ways, but two common choices are $k = 2, p \approx 2^{512}$ and $k = 6, p \approx 2^{171}$. Pairing implementations with both sets of parameters are available in the PBC library [Lyn08]. Efficiency studies suggest that $k = 2$ and the Tate pairing can offer better performance at this security level [KM05], so we make that choice.

8.2 Cost of Building a Circuit with Tor

Tor builds circuits by telescoping. A user Uriel chooses a Tor node (say Alice), and establishes a secure channel using an encrypted Diffie-Hellman exchange. She then picks a second node, Bob, and over this secure channel, establishes a new secure channel to Bob with another (end-to-end) encrypted Diffie-Hellman exchange. She proceeds in this manner until the circuit is of some desired length ℓ. For details, see the Tor specification [DM08]. Note that Uriel cannot use the same Diffie-Hellman parameters with different nodes, lest those nodes be able to determine that the same user was communicating with each of them.

Each Diffie-Hellman exchange requires Uriel to perform two modular exponentiations with 1024-bit moduli and 320-bit exponents. Likewise, each server also performs two of these exponentiations. Uriel RSA-encrypts the Diffie-Hellman parameter she sends to the server, and the server decrypts it. The AES and hashing operations involved have negligible costs compared to these.

Uriel's circuit construction to Alice takes two messages: one from Uriel to Alice, and one from Alice to Uriel. When Uriel extends this circuit to Bob (via Alice), there are four additional messages: Uriel to Alice, Alice to Bob, Bob to Alice, and Alice to Uriel. Continuing in this way, we see that the total number of messages required for Tor to construct a circuit of length ℓ is $\ell(\ell + 1)$. Note that each of these messages needs to be encrypted and decrypted at each hop.

8.3 Cost of Building a Circuit with Paring-Based Onion Routing

In order to create a circuit of length ℓ with our single-pass circuit construction, the user Uriel must choose ℓ random elements r_i of $\mathbb{Z}_n^*$. As above, Uriel should not reuse these values. She then

Table 1: Comparison of costs of setting up a circuit of length ℓ. The values in the Tor column are based on the Tor specification [DM08]. PB-OR represents our pairing-based onion routing schemes. efs-OR indicates eventual forward secret ORs, while ifs-OR indicates immediate forward secret ones. Λ_{avg} represents the average of the indices of the λ immediate forward secret nodes.

Operation	Time	Tor		PB-OR		λ-Pass PB-OR		
	(ms)	user	OR	user	OR	user	efs-OR	ifs-OR
Pairing	2.9	0	0	0	1	λ	1	1
RSA decryption	2.7	0	1	0	0	0	0	0
Exponentiation (Tor)	1.5	2ℓ	2	0	0	0	0	0
Multiplication in $\mathbb{G}$	1.0	0	0	ℓ	0	ℓ	0	1
Exponentiation in $\mathbb{G}_T$	0.2	0	0	ℓ	0	ℓ	0	1
RSA encryption	0.1	ℓ	0	0	0	0	0	0
Total time (ms)		3.1ℓ	5.7	1.2ℓ	2.9	$1.2\ell + 2.9\lambda$	2.9	4.1
Total AES-encrypted messages		$\ell(\ell+1)$		2ℓ		$2\lambda\Lambda_{\text{avg}}$		

computes $r_S U$ and $\gamma_S{}^{r_S}$, and derives the forward and backward keys K_{US} and K_{SU} from $\gamma_S{}^{r_S}$, for each server S in the circuit. Note that the γ_S values were precomputed, and cost nothing during each circuit creation. Each server computes $e(r_S U, d_S) = \gamma_S{}^{r_S}$ for its current private key d_S and derives K_{US} and K_{SU}.

Uriel creates one message, as in Figure 1, and sends it to the first server in the chain. This server decrypts a layer and sends the result to the second server in the chain, and so on, for a total of ℓ hop-by-hop encrypted messages. At the end of the chain, the last server replies with a confirmation message that travels back through the chain, producing ℓ more messages, for a total of 2ℓ.

8.4 Cost of Building a λ-Pass Paring-Based Onion Routing Circuit

Our λ-pass paring-based onion routing circuit construction is similar to that of our single-pass construction. Additional tasks that the immediate forward secret nodes must do are generation of a random integer r_{U_i}, computation of the pseudonym $r_{U_i} Q_{v\Lambda_i}$, and computation of $\gamma_{v\Lambda_i}^{r_{\Lambda_i} r_{U_i}}$. Uriel correspondingly has to perform λ pairing computations to generate modified session keys using the received pseudonyms $r_{U_i} Q_{v\Lambda_i}$ from λ immediate forward secret nodes. The number of messages and corresponding AES encryptions depends on the positions of the λ immediate forward secret nodes in the circuit. It is equal to $2\sum_{i=1}^{\lambda} \Lambda_i = 2\lambda\Lambda_{\text{avg}}$, where Λ_{avg} is the average of the indicies of the immediate forward secret nodes in the circuit.

8.5 Comparison and Discussion

We summarize the results of the previous three sections in Table 1. We count the number of "bignum" operations for each of the client and the servers, both for Tor and for our pairing-based onion routing protocols. We ignore the comparatively negligible computational costs of AES operations and hashing. For each bignum operation, we include a benchmark timing. These timings were gathered on a 3.0 GHz Pentium D desktop using the PBC pairing-based cryptography library [Lyn08].

We can see that the total computation time to construct a circuit of length ℓ using our single-pass method is 61% less on the user side and 49% less on the OR side as compared to using Tor. In addition, this circuit construction uses only a linear number of AES-encrypted messages, while Tor uses a quadratic number. As compared to single-pass circuit construction, our λ-pass circuit construction requires an additional λ pairing computations by the user, requiring a total of $1.2\ell + 2.9\lambda$ ms, and on average $2.9 + 1.2\lambda/\ell$ ms for each of the ORs. For proposed values of $\lambda = 3, 4$, or 5, these are certainly reasonable times, considering the advantage of immediate forward secrecy, and having PKVP = MKVP.

9 Conclusion

We have presented new pairing-based approaches for circuit construction in onion routing anonymity networks. We first extended the protocol of Sakai et al. [SOK00] to allow for one-way or two-way anonymous or pseudonymous key agreement. We then used this extension to produce new circuit construction protocols.

Our single pass circuit construction uses significantly less computation and communication than the corresponding protocol in Tor, and reduces the load on the network support infrastructure. To achieve immediate forward secrecy instead of eventual forward secrecy, we have also defined λ-pass circuit construction. These improvements can be used to improve the efficiency and to enhance the scalability of low-latency anonymity networks.

Acknowledgements.

We would like to thank the Natural Sciences and Engineering Research Council of Canada for supporting this research with a Discovery Grant (Kate, Goldberg) and a PGS-D postgraduate scholarship (Zaverucha). We are also grateful for funding provided in the form of David R. Cheriton Graduate Scholarships (Kate, Zaverucha). We thank the anonymous reviewers of PETS 2007 for their constructive feedback, which improved an earlier version of this paper. Finally, we would also like to thank Sk. Md. Mizanur Rahman for providing us with an advance copy of the proceedings version of [RIO$^+$06].

References

[BF01] D. Boneh and M. Franklin, *Identity-Based Encryption from the Weil Pairing*, Advances in Cryptology—CRYPTO 2001, 2001, pp. 213–229.

[Bla05] I. Blake (ed.), *Advances in elliptic curve cryptography*, London Mathematical Society Lecture Note Series, no. 317, Cambridge University Press, 2005.

[Can01] R. Canetti, *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001), 2001, pp. 136–145.

[Cha81] D. Chaum, *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*, Communications of the ACM **4** (1981), no. 2, 84–88.

[CL05] J. Camenisch and A. Lysyanskaya, *A Formal Treatment of Onion Routing*, Advances in Cryptology—CRYPTO 2005, 2005, pp. 169–187.

[CL06] H. Chien and R. Lin, *Identity-based Key Agreement Protocol for Mobile Ad-hoc Networks Using Bilinear Pairing*, IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'06), 2006, pp. 520–529.

[Dai98] W. Dai, *PipeNet 1.1*, Post to Cypherpunks mailing list, November 1998.

[DE06] R. Dupont and A. Enge, *Provably secure non-interactive key distribution based on pairings*, Discrete Applied Mathematics **154** (2006), no. 2, 270–276.

[DM08] R. Dingledine and N. Mathewson, *The Tor Protocol Specification*, `http://tor.eff.org/svn/trunk/doc/spec/tor-spec.txt`, 2008, Accessed February 2008.

[DMS04] R. Dingledine, N. Mathewson, and P. Syverson, *Tor: The Second-Generation Onion Router*, Proceedings of the 13th USENIX Security Symposium, 2004.

[DR06] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol (Version 1.1), Request for Comments (RFC) 4346*, `http://www.ietf.org/rfc/rfc4346.txt`, April 2006.

[FM02] M. J. Freedman and R. Morris, *Tarzan: A Peer-to-Peer Anonymizing Network Layer*, Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002), 2002.

[Gol06] I. Goldberg, *On the Security of the Tor Authentication Protocol*, Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006), June 2006, pp. 316–331.

[GRS96] D. Goldschlag, M. Reed, and P. Syverson, *Hiding Routing Information*, Proceedings of Information Hiding: First International Workshop, 1996, pp. 137–150.

[Hua07] D. Huang, *Pseudonym-based cryptography for anonymous communications in mobile ad hoc networks*, International Journal of Security and Networks **2** (2007), no. 3–4, 272–283.

[KG07] A. Kate and I. Goldberg, *A Distributed Private-key Generator for Identity-Based Cryptography*, Tech. Report CACR 2007-33, Centre for Applied Cryptographic Research, 2007, Available at `http://www.cacr.math.uwaterloo.ca/techreports/2007/cacr2007-33.pdf`.

[KKA03] A. Khalili, J. Katz, and W. Arbaugh, *Toward Secure Key Distribution in Truly Ad-Hoc Networks*, IEEE Workshop on Security and Assurance in Ad-Hoc Networks 2003, 2003, pp. 342–346.

[KM05] N. Koblitz and A. Menezes, *Pairing-Based Cryptography at High Security Levels*, 10th IMA International Conference on Cryptography and Coding, 2005, pp. 13–36.

[KZG07] A. Kate, G. M. Zaverucha, and I. Goldberg, *Pairing-Based Onion Routing*, 7th Privacy Enhancing Technologies Symposium (PETS 2007), 2007.

[KZH07] A. Kate, G. M. Zaverucha, and U. Hengartner, *Anonymity and Security in Delay Tolerant Networks*, 3rd International Conference on Security and Privacy in Communication Networks (SecureComm 2007), 2007.

[Lyn08] B. Lynn, *PBC Library – The Pairing-Based Cryptography Library*, `http://crypto.stanford.edu/pbc/`, 2008, Accessed February 2008.

[Möl03] B. Möller, *Provably Secure Public-Key Encryption for Length-Preserving Chaumian Mixes*, CT-RSA 2003, 2003.

[MOV91] A. Menezes, T. Okamoto, and S. Vanstone, *Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field*, 23rd annual ACM Symposium on Theory of Computing (STOC 1991), 1991, pp. 80–89.

[MOV97] A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of applied cryptography*, 1st ed., CRC Press, 1997.

[MVdV04] S. Mauw, J. Verschuren, and E. de Vink, *A Formalization of Anonymity and Onion Routing*, ESORICS 2004, 2004, pp. 109–124.

[OO05] E. Okamoto and T. Okamoto, *Cryptosystems Based on Elliptic Curve Pairing*, Modeling Decisions for Artificial Intelligence—MDAI 2005, 2005, pp. 13–23.

[ØS07] L. Øverlier and P. Syverson, *Improving efficiency and simplicity of Tor circuit establishment and hidden services*, 7th Privacy Enhancing Technologies Symposium (PETS 2007), 2007.

[RIO$^+$06] S. Rahman, A. Inomata, T. Okamoto, M. Mambo, and E. Okamoto, *Anonymous Secure Communication in Wireless Mobile Ad-hoc Networks*, First International Conference on Ubiquitous Convergence Technology (ICUCT2006), 2006.

[RP02] M. Rennhard and B. Plattner, *Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection*, Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002), 2002.

[RSG98] M. Reed, P. Syverson, and D. Goldschlag, *Anonymous Connections and Onion Routing*, IEEE Journal on Selected Areas in Communications **16** (1998), no. 4, 482–494.

[Sha79] A. Shamir, *How to Share a Secret*, Commun. ACM **22** (1979), no. 11, 612–613.

[SK05] A. Seth and S. Keshav, *Practical Security for Disconnected Nodes*, IEEE ICNP Workshop on Secure Network Protocols (NPSec 2005), 2005, pp. 31–36.

[SOK00] R. Sakai, K. Ohgishi, and M. Kasahara, *Cryptosystems based on pairing*, Symposium on Cryptography and Information Security (SCIS 2000), 2000.

[STRL00] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, *Towards an Analysis of Onion Routing Security*, Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability, 2000, pp. 96–114.

[The08] The Tor Project, *Tor: anonymity online*, `http://tor.eff.org/`, 2008, Accessed February 2008.

[Ver01] E. Verheul, *Evidence that XTR Is More Secure than Supersingular Elliptic Curve Cryptosystems*, Advances in Cryptology—Eurocrypt 2001, 2001, pp. 195–210.