Provably Secure and Practical Onion Routing

Michael Backes
Saarland University and MPI-SWS, Germany
backes@cs.uni-saarland.de

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

Aniket Kate
MPI-SWS, Germany
aniket@mpi-sws.org

Esfandiar Mohammadi
Saarland University, Germany
mohammadi@cs.uni-saarland.de

Abstract

The onion routing network, Tor, is undoubtedly the most widely employed technology for anonymous web access. Although the underlying onion routing (OR) protocol's multi-pass cryptographic circuit construction appears satisfactory, a comprehensive formal analysis of its security guarantees is still lacking. Moreover, in practice the current Tor circuit construction suffers from inefficiency, which is due to the key exchange protocol that is used for circuit construction. Consequently, significant efforts have been put towards improving the efficiency of the key exchange in onion routing.

In this paper, we address both these issues. We present the first security definition for OR protocols with multi-pass circuit construction in the universal composability framework. We then show that a recently introduced efficient key exchange protocol can be used in the circuit construction such that the resulting OR protocol provably satisfies our security definition. As a result, we obtain the first provably secure and practical OR protocol with multi-pass circuit construction.

Contents

1 Introduction

Over the last few years the onion routing network, Tor [21], has emerged as a successful technology for anonymous web browsing. It is currently serving hundreds of thousands of users across the world, and it is employing more than two thousand dedicated relays. Its impact is also evident from the media coverage it has received over the last few years [14]. Despite its success, the existing Tor network still lacks a rigorous security analysis, as its security properties have neither been formalized cryptographically nor proven. (See [3,9,19] for the previous attempts and their shortcomings.) In this paper, we define security for the second-generation onion routing (OR) protocol Tor, and construct a provably secure and practical OR protocol.

An OR network consists of a set of routers or OR nodes that relay traffic, a large set of users, and directory servers that provide to the users routing information of the OR nodes. A user constructs a *circuit* by choosing a small sequence of (usually three) OR nodes, where the chosen nodes route the user's traffic over the path formed. The crucial property of an OR protocol is that no node in a circuit can determine other circuit nodes than its predecessor and its successor. The user achieves this by sending the first OR node a message wrapped in multiple layers of symmetric encryption (one layer per node), called an *onion*, using symmetric keys agreed upon during an initial *circuit construction* phase. Consequently, given a public-key infrastructure (PKI), the cryptographic challenge in onion routing is to agree upon such symmetric keys.

In the first generation OR circuit construction [22], the challenge of agreeing on a key was met by a user including the identifier of the next node and a random symmetric session key in each onion layer that is encrypted with the node's public key. However, such a *single-pass* circuit construction cannot be *forward secret*: if an adversary corrupts a node and obtains its private key, then the adversary can decrypt all of the node's past communication. Although changing the public/private key pairs for all OR nodes after a predefined interval is a possible solution (*eventual* forward secrecy), this solution does not scale to realistic OR networks such as Tor, since at each interval start every user has to download a new set of public keys for all the nodes.

The scalability issues with single-pass OR networks pursuing forward secrecy have been addressed by Dingledine, Mathewson and Syverson [7]. They introduced a telescoping approach for the second generation OR protocol Tor. In this telescoping approach, they employed a *multi-pass* key agreement protocol called the Tor authentication protocol (TAP) to negotiate a symmetric session key between a user and a node. Here, the node's public key is only used to initiate the construction, and the compromise of this public key does not invalidate the secrecy of the session keys once the randomness used in the protocol is erased.

Goldberg [10] presented a security proof for TAP in the random oracle model. The security of TAP, however, does not automatically imply the security of the Tor circuit construction, since the Tor circuit construction constitutes a sequential execution of multiple TAP instances. Therefore, the security of the OR circuit construction has to be analyzed in a composability setting. In this direction, Camenisch and Lysyanskaya [3] defined an anonymous message transmission protocol in the universal composability (UC) framework. They motivated their choice of the UC framework for a security definition by its versatility as well as the lack of other appropriate models that capture protocol compositions. They also went ahead and presented a construction of a protocol that satisfies their definition. However, Feigenbaum, Johnson and Syverson [9] observe that the protocol definition presented by Camenisch and Lysyanskaya [3] does not correspond to the OR methodology. In particular, in the work of Camenisch and Lysyanskaya a circuit is not constructed before messages are transmitted. Therefore, a rigorous security analysis of an OR protocol is still missing. We present an OR definition and a provably secure OR construction that accurately models the OR methodology used in practice.

Our Contributions. We first present a security definition of OR protocols in the UC framework. We then cryptographically characterize the security of the modules that centrally underlie OR: a one-way authenticated key exchange (1W-AKE) primitive, and onion forming, wrapping and unwrapping algorithms. Next, for a given a set of secure OR modules, we present a general construction of a secure OR protocol. Finally, we present a practical OR protocol using the following OR modules: a 1W-AKE,

ntor [11][1], and a variant of the onion forming, wrapping, and unwrapping algorithms introduced in the work of Camenisch and Lysyanskaya [3].

We refine previous work [3] in multiple respects. First and most importantly, we construct an OR circuit interactively in multiple passes, whereas in the previous work no circuit is constructed, which does not model the widely used Tor protocol. Their approach, and even single-pass circuit construction in general, restricts the protocol to eventual forward secrecy, while a multi-pass circuit construction allows for ensuring forward secrecy immediately after the circuit is closed. Second, the security definition of Camenisch and Lysyanskaya prohibits the adversary from re-wrapping onions, a constraint for which the authors do not give a practical motivation. In our construction, we remove this restriction, leading to more relaxed security requirements. Finally, they assume that the adversary is static, i.e., it has to specify all corrupt nodes beforehand, while our model guarantees composable security against attackers that may corrupt long-term keys adaptively.

Outline. The paper is organized as follows: Section 2 provides background information relevant to onion routing, one-way authenticated key exchange, and the UC framework. In Section 3, we present our security definition for onion routing. In Section 4, we present a cryptographic definition for secure OR modules, i.e., for a 1W-AKE primitive, and for onion forming, wrapping, and unwrapping algorithms. In Section 5, we show that, given a set of secure OR modules, we can construct a secure OR protocol. Finally, in Section 6, we present an efficient construction for secure OR modules. We conclude and discuss further interesting directions for future work in Section 7.

2 Background

In this section, we provide background information about onion routing circuit construction, one-way authenticated key exchange, and the UC framework.

2.1 Onion Routing Circuit Construction

In the original Onion Routing project [12, 13, 22, 24], circuits were constructed in a single pass. A user chooses a path of OR nodes to a receiver, and creates a *forward onion* with several layers. Each onion layer is targeted at one node in the path and is encrypted with that node's public key. A layer contains that node's symmetric session key for the circuit, the next node in the path, and the next layer. Each node decrypts a layer using its secret key, stores the symmetric key, and forwards the next layer of the onion along to the next node. Once the last node in the path, i.e., the receiver, gets its symmetric session key, it responds with a confirmation message encrypted with its session key. Every node in the path wraps (encrypts) the *backward onion* using its session key in the reverse order, and the message finally reaches the user. A circuit that is constructed in this way, i.e., the sequence of established session keys, is thereafter used for constructing and sending onions via this circuit.

Unfortunately, there is a scalability issue in a single-pass circuit construction when pursuing forward secrecy: the forward secrecy relies on the public keys being replaced and distributed regularly. There are attempts to solve this scalability issue. Kate, Zaverucha and Goldberg [17] suggested the use of an identity-based cryptography (IBC) setting and defined a pairing-based onion routing (PB-OR) protocol. Catalano, Fiore and Gennaro [5] suggested the use of a certificateless cryptography (CLC) setting [1] instead and defined two certificateless onion routing protocols (CL-OR and 2-CL-OR). However, both of these approaches do not yield satisfactory solutions: CL-OR and 2-CL-OR suffer from the same scalability issues as that of the original OR protocol [16], while PB-OR systems raise issues in implementing a distributed private-key generator [15].

Another problem with the single-pass approach is that so far it only yields systems that guarantee *eventual* forward secrecy, i.e., if the private key is leaked only those past sessions remain secret that used an expired public key. A desirable property is that all past sessions that are closed remain secret even if the private key is leaked; such a property is called *immediate* forward secrecy. It has been shown to be impossible to obtain immediate forward secrecy with any single-pass construction [18, Sec. 5.1].

[1] Goldberg, Stebila, and Ustaoglu [11] present an attack on and a fix for the fourth protocol in work of Øverlier and Syverson [20]; the ntor protocol is the fixed protocol.

Therefore, in the current Tor protocol, circuits are constructed using a multi-pass approach that is based on an authenticated key agreement (using a public-key infrastructure). The idea is to use the private key only for establishing a temporary session key in a key exchange protocol. Together with the private key additional temporary (random) values are used for establishing the key such that mere knowledge of the private key does not suffice for reconstructing the session key. These temporary values are erased immediately after the session key has been computed. In this way it is possible to achieve immediate forward secrecy in a multi-pass circuit construction.

Although the multi-pass approach incurs an additional overhead, in practical almost all Tor circuits are constructed for a circuit length of $\ell = 3$. Therefore the multi-pass approach merely causes an overhead of six additional messages[2]. Nonetheless, despite the additional overhead, the multi-pass circuit construction looks to be the preferred choice in practice, due to its improved forward secrecy guarantees. Consequently, for our provably secure onion routing protocol we consider a multi-pass circuit construction, using a public-key infrastructure.

2.2 One-Way Authenticated Key Exchange

In a multi-pass circuit construction, a session key is established via a Diffie-Hellman (DH) key exchange. However, the precise properties required of this protocol were not formalized until recently. Goldberg, Stebila and Ustaoglu [11] formalized the concept of one-way authenticated key exchange (1W-AKE), presented an efficient instantiation, and described its utility towards onion routing. We overview their results here and we refer the readers to [11] for a detailed description.

An authenticated key exchange (AKE) protocol establishes an authenticated and confidential communication channel between two parties. Although AKE protocols in general aim for key secrecy and mutual authentication, there are many practical scenarios such as onion routing where mutual authentication is undesirable. In such scenarios, two parties establish a private shared session key, but only one party authenticates to the other. In fact, the unauthenticated party may even want to preserve its anonymity, e.g., in Tor. In their 1W-AKE protocol, Goldberg et al. formalize this precise primitive. They show that an authenticated key exchange protocol suggested for Tor — the fourth protocol in [20] — can be attacked, leading to an adversary determining all of the user's session keys. They then fixed the protocol (see Figure 6) and proved that the fixed protocol satisfies the formal properties of 1W-AKE. In our formal analysis of onion routing, we use their formal definition and their fixed protocol and show that the resultant OR protocol is UC secure.

2.3 The UC Framework: An Overview

The UC framework is designed to enable a modular analysis of large-scale security protocols. The framework establishes the notion of universal composability that characterizes which protocols remain secure under arbitrary composition with other protocols, in particular in the presence of several concurrently running instances of the same protocol. In the UC framework the security goal of a protocol is specified by a setting in which the protocol is replaced by a trusted machine, called the *ideal functionality*, to whom every party has a private channel. Such an ideal functionality locally computes any function that the real protocol would compute and specifies the information leaked to, and the capabilities of, an attacker. As an example, consider a protocol that establishes a secret channel: an attacker controlling the network can see the length of the message and prevent the message from reaching its destination. The ideal functionality for such a secret channel protocol leaks nothing more than the length of the message to the attacker and gives the attacker no more interaction capabilities than control over the delivery of messages.

We say that a protocol π *UC-realizes* an ideal functionality $\mathcal{F}$ if for all attackers A there is a simulator S such that no probabilistic poly-time (ppt) machine can distinguish an interaction with π and A from an interaction with $\mathcal{F}$ and S. The universal composability is achieved by requiring that the distinguisher, which can be thought of as the environment, is actually allowed to interact with the protocol and the attacker (or the simulator). This environment has a direct channel to the protocol parties, provides the parties with protocol inputs, and receives from the parties the designated protocol outputs; e.g., in a secret channel protocol, the message to be sent. More formally, a protocol π is said to UC-realize an ideal

[2]The overhead reduces to four additional messages, if we consider the "CREATE_FAST" option available in Tor.

functionality $\mathcal{F}$ if for all attackers there is a simulator such that for all environments the interaction with π and the attacker is indistinguishable from the interaction with $\mathcal{F}$ and simulator. All these machines are probabilistic polynomial-time interactive Turing machines.

We stress that the UC framework does not provide a notion of time; hence, the analysis of timing attacks (such as traffic analysis) is outside of the scope of this work.

3 Security Definition of OR

We start our discussion by describing the attacker model and the setup assumptions. We then abstract onion routing as a black box, more precisely as an ideal functionality $\mathcal{F}_{\text{OR}}$ in the UC framework.

3.1 System Setting

We consider a fully connected network of n parties $P_1, \ldots, P_n$. For simplicity of presentation we consider all parties to be OR nodes, i.e., relays, that also initiate circuits and send messages. It is also possible to use our formulation of onion routing for modelling users, i.e., clients that do not relay onions, by considering parties that only send and respond to messages but not relay onions.

Recall that the UC framework does not provide a notion of time. We model the expiration time of a circuit by bounding the number of times that a circuit can be used before expiring. This bound is denoted as *circuit_ttl*.

We assume that the attacker controls the network and can cause long-term keys of protocol parties to be revealed adaptively. An attacker that controls the network is too strong for most practical purposes and can simply break the anonymity of an OR protocol by holding back all but one onion and tracing that one onion though the network. Therefore, it might be more accurate to consider local attackers that control single nodes but not the entire network. How to model the attacker, however, is orthogonal to our result, as our ideal functionality also allows such local attacks.

Notation. In the sequel, we often omit the security parameter κ when calling an algorithm A, i.e., instead of writing $A(1^\kappa, x)$ (x being some argument) we write $A(x)$. Moreover, we write $y \leftarrow A(x)$ for the result y of a randomized algorithm A on input x. In contrast, we write $y := A(x)$ for the result y of a (deterministic) algorithm A on input x.

We assume that the message space is fixed for a given security parameter. This message space is denoted as $M(\kappa)$.

3.2 Ideal Functionality: OR as a Black Box

We motivate and present an ideal functionality $\mathcal{F}_{\text{OR}}$ for onion routing with a multi-pass circuit construction. We prove in Section 2.1 that this ideal functionality can serve as a black box for an OR protocol, e.g., in the analysis of anonymity services that use OR protocols.

The ideal functionality $\mathcal{F}_{\text{OR}}$, presented in Figure 1, sends to an attacker the information that is inherently leaked in an OR protocol: if an onion is sent between two parties, only a fresh handle h is sent to the attacker. Moreover, $\mathcal{F}_{\text{OR}}$ grants the attacker the capabilities that are unpreventable for an attacker that controls the network: first, the attacker can establish a circuit and send a message m to a party P. Such an action is abstracted via the command $(\mathsf{attacker\_message}, m, P)$. We stress that $\mathcal{F}_{\text{OR}}$ does not need to reflect reroutings and circuit establishments initiated by the attacker, because the attacker learns, loosely speaking, no new information by rerouting onions.[3] Second, we have to model that a long-term key is leaked. This is modelled by a command $(\mathsf{reveal\_longterm\_keys}, P)$. Upon such a command, the attacker impersonates the party P, and all future sessions *sid* of P are marked as $\mathit{malicious}(\mathit{sid}) = \mathit{true}$, since the attacker learns the key. We stress that only future sessions are corrupted; hence, our ideal functionality captures the notion of immediate forward secrecy of the OR protocol.

The ideal functionality represents a circuit as a sequence $(P_0, (P_j, \mathit{sid}_j)_{j=1}^{\ell})$ of an initiating party P_0, router parties P_j, and corresponding session identifiers sid_j. An onion $(P_0, (P_j, \mathit{sid}_j)_{j=1}^{\ell}, m, i)$ is represented as a circuit $(P_0, (P_j, \mathit{sid}_j)_{j=1}^{\ell})$, a message m, and an index i that marks the next receiving

[3] More formally, the simulator can compute all responses for rerouting or such circuit establishments without requesting information from $\mathcal{F}_{\text{OR}}$ (as shown in the proof of Theorem 1).

Upon receiving an input setup for party P:
Send setup over the network. Upon the response ok from the network, answer ok to party P.

Upon receiving an input $(\mathsf{send}, \mathcal{P}, m)$ for party P:
Do nothing if $m \notin M(\kappa)$. Otherwise, if $\mathit{established}(P, \mathcal{P}) = \emptyset$ and $|\mathcal{P}| < \mathit{circuit\_ttl} + 1$, call $\mathit{Extend\_Circuit}(P, \emptyset, \mathcal{P}, m)$. If $\mathit{established}(P, \mathcal{P}) \neq \emptyset$, let $(P, (P_1, sid_1), \ldots, (P_\ell, sid_\ell)) = \mathcal{C} = \mathit{established}(P, \mathcal{P})$. If $\ell > 0$ and $\mathit{Used}(\mathcal{C}) < \mathit{circuit\_ttl}$, call $\mathit{Send\_Message}(\mathcal{C}, m, P_1)$. If $\mathit{Used}(\mathcal{C}') \geq \mathit{circuit\_ttl}$, set $\mathit{established}(P, \mathcal{P}) := \emptyset$ and call $\mathit{Extend\_Circuit}(P, \emptyset, \mathcal{P}, m)$.

Upon receiving an input $(\mathsf{respond}, sid, m)$ for party P:
Do nothing if $m \notin M(\kappa)$. If $\bot \neq \mathit{sparties}(sid) = (P', P)$, look the circuit $\mathcal{C}$ up that ends with the session sid, and call $\mathit{Send\_Onion}((\mathcal{C}, m, \ell), P', \mathsf{back})$ (ℓ being the length of the circuit $\mathcal{C}$).

Upon receiving a message $(\mathsf{deliver}, h, P_1, P)$ from the network such that $\bot \neq \mathit{pending}(h)$:

1. *(m is a forward onion)* If $\mathit{pending}(h) = ((P_0, (P_j, sid_j)_{j=1}^{\ell}), m', i, P_i)$ and $(P_i, P_{i+1}) = \mathit{sparties}(sid)$ distinguish three cases. Let ℓ be the length of the circuit $\mathcal{C}$, $P := P_i$, and $sid := sid_i$.
   (i.) If $i \geq \ell$ and $m' = (\mathsf{initiate}, sid', P_2)$, set $\mathit{pcandidate}(P, sid) := sid'$. If the long-term key of P has been revealed, set $\mathit{malicious}(sid') := \mathit{true}$, draw a fresh handle h, store $\mathit{pending}(h) := (\mathcal{C}, m', i+1, P_{i+1})$, and send (h, m', sid', P, P_2) over the network. If P_2's long-term key has been revealed, send (h, m', sid', P, P_2); otherwise, merely send (h, P, P_2) over the network.
   (ii.) If $i \geq \ell$ and $m' \neq (\mathsf{initiate}, sid', P_2)$, output $(\mathsf{result}, m', sid)$ to P.
   (iii.) If $i < \ell$ and (P_2, sid') the ith element of $\mathcal{C}$, call $\mathit{Send\_Onion}((\mathcal{C}, m', i+1), P_2, \mathsf{forw})$.
2. *(m is a backward onion)* If $\mathit{pending}(h) = (P_0, (P_j, sid_j)_{j=1}^{\ell}, m', i)$ and $\mathit{sparties}(P_i, sid_i) = (P_i, P_{i+1})$, let $P := P_i$ and $sid := sid_i$ and distinguish two cases.
   (i.) If $i = 0$ and $\mathit{predecessor}(P, sid) = \bot$ and $\bot \neq ((P, (P_1, sid), \ldots, (P_k, sid_k)), P_{k+1}, \mathcal{P}, m) = \mathit{open\_AKE}(sid)$ (in case $k = 0$ take Ψ instead of $sid = (\Psi, \Psi')$). If $m' = (\mathsf{confirm}, P_{k+1}, sid_{k+1})$, store $\mathit{established}(P, P_1, \ldots, P_{k+1}) := ((P_1, sid_1), \ldots, (P_{k+1}, sid_{k+1}))$ and $\mathit{sparties}(sid_{k+1}) := (P, P_{k+1})$ and set $\mathit{open\_AKE}(sid) := \bot$. If $k + 1 < |\mathcal{P}|$, call $\mathit{Extend\_Circuit}(P, (P_1, \ldots, P_{k+1}), \mathcal{P}, m)$; otherwise, call $\mathit{Send\_Message}(\mathcal{C}, sid, m, P_1)$.
   (ii.) If $\mathit{predecessor}(P, sid) = \bot$ and $\mathit{open\_AKE}(sid) = \bot$, output (result, m') to P.
   (iii.) *(m has to be relayed)* If $sid' = \mathit{predecessor}(P, sid)$ and $\bot \neq \mathit{sparties}(sid') = (P_2, P)$, call $\mathit{Send\_Onion}(((P_2, sid'), \mathcal{C}', m'), P_2, \mathsf{back})$.
3. *(m is the response from a key exchange)* If $\mathit{pending}(h) = m$, $m = (\mathsf{confirm}, P_1, sid)$, $sid = (\Psi', \Psi)$ and $sid' = \mathit{pcandidate}(\Psi') \neq \bot$, set $\mathit{predecessor}(sid) := sid'$. If $\bot \neq \mathit{sparties}(sid') = (P_2, P)$, look the circuit $\mathcal{C}$ up that ends with sid' and call $\mathit{Send\_Onion}((\mathcal{C}, m), P_2, \mathsf{back})$.
4. *(m is the initial message from a key exchange)* If $\mathit{pending}(h) = m$, $m = (\mathsf{initiate}, P, sid)$, set $\mathit{sparties}(sid, P) := (P_1, P)$, and draw a fresh handle h, set $m' := (\mathsf{confirm}, P_1, sid)$, $\mathit{pending}(h) := (m', sid, P, P_1)$, and send (h, P, P_1) over the network if P_1's long-term key has not yet been revealed, and send (h, m', P, P_1) otherwise.

Figure 1: The ideal functionality $\mathcal{F}_{\text{OR}}$ for onion routing

party P_i for which the onion is meant. The ideal functionality $\mathcal{F}_{\text{OR}}$ expects two kinds of inputs: either a send command consisting of a message m to be sent and a route, or a $\mathsf{respond}$ command consisting of a response message m and a session identifier sid, referring to the circuit for the response. Upon receiving a send command $(\mathsf{send}, \mathcal{P}, m)$ for party P with a message m and a path $\mathcal{P} = (P_1, \ldots, P_\ell)$, $\mathcal{F}_{\text{OR}}$ checks whether there is a valid established circuit from P over path. If not, $\mathcal{F}_{\text{OR}}$ establishes a new circuit (see subroutine *Extend_Circuit*). Given a circuit $(P, (P_j, sid_j)_{j=1}^{\ell})$ and a message m, $\mathcal{F}_{\text{OR}}$ constructs an onion $(P, (P_j, sid_j)_{j=1}^{\ell}, m, 1)$ and sends $(h, |\mathcal{P}|, |m|, P, P_i)$ to the adversary for a freshly drawn handle h, which is executed by a call to the subroutine *Send_Onion*. Upon receiving a $\mathsf{respond}$ command $(\mathsf{respond}, m, sid, P)$, $\mathcal{F}_{\text{OR}}$ looks up the circuit $\mathcal{C}$ that ends with the session sid, and constructs an onion $(\mathcal{C}, m, \ell')$ (ℓ' being the length of the path), and sends a fresh handle of the onion calling the subroutine *Send_Onion*.

How to process network messages. $\mathcal{F}_{\text{OR}}$ expects from the attacker for every network message, i.e., for every handle h, the permission to deliver this handle. For a handle delivery permission $(\mathsf{deliver}, h, P_1, P)$ in which the handle h corresponds to a forward onion, we distinguish three cases: In the first case, the onion has only one layer left and the message signalizes a circuit extension. Then, a freshly drawn handle and the party identities are sent over the network, i.e., (h, P, P') (for a receiving

Extend_Circuit$(P, (P_j)_{j=1}^{k}, (P_j)_{j=1}^{\ell}, m)$: Let $((P_1, sid_1), \ldots, (P_k, sid_k)) := established(P, \mathcal{P})$ (where $k = 0$ for $\mathcal{P}' = \emptyset$). Draw a fresh session id sid_{k+1} and let $m' := (\mathsf{initiate}, P_{k+1}, sid_{k+1})$. Store $(P, established(P, \mathcal{P}), (P_{k+1}, sid_{k+1}), \mathcal{P}, m')$ in $open\_AKE(sid_1)$. Then, call $Send\_Message((P, established(P, \mathcal{P}), m'), P_{k+1})$.

Send_Message$((P, (P_j, sid_j)_{j=1}^{k}, m), P')$:

1. If $n = 0$ (i.e., the circuit is empty), draw a fresh handle h, store $pending(h) := ((P, \emptyset), m, 1, P')$, and send $(h, 0, |m|, P, P')$ over the network.
2. If $\mathcal{C} \neq \emptyset$, increase $Used(\mathcal{C})$ and call $Send\_Onion((P, (P_j, sid_j)_{j=1}^{k}, m, 1), P', \mathsf{forw})$.

Send_Onion$((\mathcal{C}, m, i), P', dir)$: Draw a fresh handle h, set $pending(h) := (\mathcal{C}, m, i, P')$. Let $\mathcal{C} = (P, (P_j, sid_j)_{j=1}^{\ell})$ and distinguish two cases.

1. If $dir = \mathsf{forw}$, let $\mathcal{Q}$ be the longest contiguous malicious sequence beginning from (P_{i+1}, sid_{i+1}), i.e, $(P_{i+1}, sid_{i+1}), \ldots, (P_s, sid_s)$ such that $malicious(sid_{i+1}) = \cdots = malicious(sid_s) = true$. If $\mathcal{Q}$ does not contain the last party of $\mathcal{C}$, i.e., $s < \ell$, let P_{s+1} be the first party corresponding to the first honest session sid_{s+1} ($malicious(sid_{s+1} \neq true)$) after $\mathcal{Q}$, and send either $(h, |\mathcal{C}|, |m|, P, \mathcal{Q}, sid_{s+1}, P_{s+1})$ (if P_{s+1} can be impersonated) or $(h, |\mathcal{C}|, |m|, P, \mathcal{Q}, P')$ (if P_{s+1} is honest) over the network; otherwise send $(m, \mathcal{Q})$ over the network.
2. If $dir = \mathsf{back}$, proceed as in the forward onion case except for the difference that the sequence begins from (P_{i-1}, sid_{i-1}) and stops with a party P_{s-1} corresponding to an honest session sid_{s-1}.

Figure 2: Subroutines of $\mathcal{F}_{\text{OR}}$

party P'). In the second case, the onion has only one layer left and the message is not a circuit extension. Then, $\mathcal{F}_{\text{OR}}$ outputs this message either to the party (if the party is uncorrupted) or to the attacker (if the party is corrupted). In the third case, the forward onion has more than one layer left. Then, the onion is relayed by calling the subroutine *Send_Onion*.

For a handle that corresponds to a backward onion, we also distinguish three cases: In the first case, the receiving party P is the initiator of the circuit and the circuit construction is not yet completed. Then, $\mathcal{F}_{\text{OR}}$ either extends the circuit or, if the circuit construction is completed, sends the message. In the second case, P is also the initiator of the circuit but the circuit construction has already been completed. Let m be the message that the onion carried. Then, $\mathcal{F}_{\text{OR}}$ outputs m. In the third case, P knows a predecessor to the session sid of the onion, i.e., P acts as a relay. Then, P looks the corresponding party P' for sid up, forwards the onion by wrapping it with (P', sid), and calling the subroutine *Send_Onion*.

If the message is not a handle for an onion but for a session establishment message, $\mathcal{F}_{\text{OR}}$ distinguishes two cases. In the first case, the message is a handle for the response from a key exchange $(\mathsf{confirm}, P_1, sid)$. Then, $\mathcal{F}_{\text{OR}}$ memorizes that the receiving party P knows that a previously stored candidate session sid' is in some circuit the predecessor of sid (implemented as $predecessor(P, sid) := sid'$). Moreover, $\mathcal{F}_{\text{OR}}$ also memorizes that there is a session sid between P and P_1 (implemented as $sparties(P, sid) := (P, P_1)$). In the second case, the message is the initiation of a key exchange $(\mathsf{initiate}, P, sid)$. Then, $\mathcal{F}_{\text{OR}}$ memorizes that the sender P_1 established a session sid with the receiving party P (i.e., it sets $sparties(P, sid) := (P_1, P)$).

Attacker actions in $\mathcal{F}_{\text{OR}}$. The attacker can send the following two commands.
Upon receiving a message $(\mathsf{attacker\_message}, m, P)$: Output m to the party P.

Upon receiving a message $(\mathsf{reveal\_longterm\_keys}, P)$ from the network: Remember that the long-term key of P has been revealed. For party P $\mathcal{F}_{\text{OR}}$ grants the attacker the opportunity to impersonate P.

4 Secure OR modules

For the sake of reusability, we reduce the security of a UC secure OR protocol to the security of its core cryptographic primitives (see Section 5). In this section, we present a cryptographic characterization of these core cryptographic primitives, which we call *secure OR modules*. We believe that proving the security of OR modules is significantly less effort than proving the UC security of an entire protocol. Secure OR modules mainly consist of two parts: First, a *one-way authenticated key-exchange* primitive

(1W-AKE), a notion recently introduced by Goldberg, Steblia, and Ustaoglu [11]. Second, onion forming, unwrapping, and wrapping algorithms that jointly satisfy *onion security*, a refinement of the notion introduced by Camenisch and Lysyanskaya [3].

The 1W-AKE establishes a symmetric key between two parties such that the identity of the initiator cannot be derived from the protocol messages; moreover, given a public-key infrastructure the 1W-AKE guarantees that the second party cannot be impersonated. Such a 1W-AKE consists of four algorithms G, *Initiate*, *Respond*, and *Compute_Key*. The algorithm G denotes the key generation algorithm for the long-term public and secret key of a party. *Initiate* takes as input the public key of the authenticated party and generates the initial key-exchange message m_1. *Respond* takes as input his secret key and the initial message m_1 and outputs the session key and the response message m_2. The algorithm *Compute_Key* is run on the response m_2 and the public key of the authenticated party, i.e., the responder; finally, *Compute_Key* outputs the key.

The onion forming, unwrapping, and wrapping algorithms are denoted as *FormOn*, *UnwrapOn*, and *WrapOn*. $\mathit{FormOn}(m, (P_i, k_i, \mathit{sid}_i)_{i=1}^{\ell}, \ell')$ forms an onion for the circuit $(P_i, k_i, \mathit{sid}_i)_{i=1}^{\ell}$. The additional parameter ℓ' is used for responding to a message. Since we require our construction to hide the length of the remaining path, the responder node needs to know the length of the path already when constructing the first layer. We call this ℓ' the *appearing length* of the onion. In an honest protocol run, ℓ equals ℓ'. This procedure is called by a router that, given a message m and an established circuit $k_1, \ldots, k_\ell$, is going to send m via this established circuit.[4] $\mathit{UnwrapOn}(O, k)$ peels one layer of the onion O using the session key k. This procedure is called by a router that removes a layer from an onion and forwards the inner layer to the next party in the circuit. Moreover, this procedure also determines the appearing length of the onion. The last procedure $\mathit{WrapOn}(O, P, k, \mathit{sid})$ is used in an OR protocol for responding to an anonymous message without knowing the receiver's identity. This procedure is called by a router that either responds to an anonymous message or forwards a response.

Recall that we assume a public-key infrastructure, i.e., every party knows a secret key whose corresponding public key has been distributed, and all public keys have been certified and checked. We denote by pk_P the public key of the party P and by sk_P its secret key.

One-way authenticated key-exchange. The first property that a 1W-AKE has to satisfy is correctness: if all parties behave honestly, then the protocol establishes a shared key.

Definition 1 (Correctness of 1W-AKE)**.** *Let a public-key infrastructure be given, i.e., for every party P every party knows a (certified) public key pk_P and P itself also knows the corresponding secret key sk_P. Let $AKE := (\mathit{Initiate}, \mathit{Respond}, \mathit{Compute\_Key})$ be a tuple of polynomial-time bounded randomized algorithms. We say that AKE is a* correct one-way authenticated key-agreement *if the following holds for all parties A, B:*

$$
\begin{aligned}
\Pr[\ & (\mathit{ake}, B, m_1, \Psi_A, \mathit{state}) \leftarrow \mathit{Initiate}(pk_B, B, m), \\
& ((\mathit{ake}, B, m_2, \Psi'_A, \Psi_B), (k_2, \star, \vec{v})) \\
& \qquad\qquad \leftarrow \mathit{Respond}(sk_B, B, m_1, \Psi_A), \\
& (k_1, B, \vec{v}\,') := \mathit{Compute\_Key}(pk_B, m_2, \mathit{state}, \Psi_B) \\
& : k_1 = k_2 \ \mathit{and}\ \ \vec{v} = \vec{v}\,' \\
&] = 1
\end{aligned}
$$

The 1W-AKE challenger for security. Goldberg, Stebila, and Ustaoglu [11] formalize the security of a 1W-AKE by defining a *challenger* that represents all honest parties. The attacker is then allowed to query this challenger. Loosely speaking, if the attacker is not able to distinguish a fresh session key from a randomly chosen session key we say that the 1W-AKE is *secure*.

The challenger answers the following queries of the attacker. Internally, the challenger runs the algorithms of *AKE*. All queries are directed to some party P; we denote this party in a superscript. If the party is clear from the context, we omit the superscript, e.g., we then write $\mathsf{send}(m)$ instead of $\mathsf{send}^P(m)$.

$\mathsf{send}^P(\mathit{params}, P')$: Compute $(m, \mathit{state}) \leftarrow \mathit{Initiate}(pk_P, P', \mathit{params})$. Send m to the attacker.

[4] Recall that we ensure that m is in some message space $M(\kappa)$.

$\mathsf{send}^P(\Psi, msg, P')$: If $akestate(\Psi) = \perp$ and $P' = P$, compute $(m, \mathsf{result}) \leftarrow Respond(sk_P, P, msg, \Psi)$. Otherwise, if $msg = (msg', Q)$ compute $(m, \mathsf{result}) \leftarrow Compute\_Key(pk_Q, msg', akestate(\Psi), \Psi)$. Then, send m to the attacker.

$\mathsf{reveal\_longterm\_keys}^P$ The challenger returns the long-term key of P to the attacker.

If any verification fails, i.e. one of the algorithms outputs $\perp$, then the challenger erases all session-specific information for that party and aborts the session.

Additionally, the attacker has access to the following oracle in the 1W-AKE security experiment:

$\mathsf{test}(P, \Psi)$: Abort if party P has no key stored for session Ψ or the partner for session Ψ is anonymous (i.e., P is not the initiator of session Ψ). Otherwise, choose $b \xleftarrow{\$} \{0, 1\}$. If $b = 1$, then return the session key k; otherwise, if $b = 0$, return a randomly chosen element from the key space. Only one call to test is allowed.

We say that a session Ψ at a party i is *fresh* if for no party involved in that session the long-term key has been revealed.

Definition 2 (One-way-AKE-security)**.** *Let κ be a security parameter and let $n \geq 1$. A protocol π is said to be* one-way-AKE-secure *if, for all ppt adversaries M, the advantage that M distinguishes a session key of a one-way-AKE-fresh session from a randomly chosen session key is negligible (in κ).*

The 1W-AKE challenger for one-way anonymity. For the definition of *one-way anonymity* we introduce a proxy, called the anonymity challenger, that relays all messages from and to the 1W-AKE challenger except for a challenge party C. The attacker can choose two challenge parties, out of which the anonymity challenger randomly picks one, say i^*. Then, the anonymity challenger relays all messages that are sent to C to P_{i^*} (via the 1W-AKE challenger).

In the one-way anonymity experiment, the adversary can issue the following queries to the challenger C. All other queries are simply relayed to the 1W-AKE challenger. The session Ψ^* denotes the challenge session. The two queries are for activation and communication during the test session.

$\mathsf{start}^C(i, j, params, P)$: Abort if $i = j$. Otherwise, set $i \xleftarrow{\$}$ and $(\Psi^*, msg) \leftarrow \mathsf{send}^{P_{i^*}}(params, P)$; return msg'. Only one message start^C is processed.

$\mathsf{send}^C(msg)$: Relay $\mathsf{send}^{P_{i^*}}(msg)$ to the 1W-AKE challenger. Upon receiving an answer msg', forward msg' to the attacker.

Definition 3 (One-way anonymity)**.** *Let κ be a security parameter and let $n \geq 1$. A protocol π is said to be* one-way anonymous *if, for all ppt adversaries M, the advantage that M wins the following experiment $\mathsf{Expt}^{1w-anon}_{\pi,\kappa,n}(M)$ is negligible (in κ).*

1. *Initialize parties $P_1, \ldots, P_n$.*
2. *The attacker M interacts with the anonymity challenger, finishing with a message $(\mathsf{guess}, \hat{i})$.*
3. *Suppose that M made a $\mathsf{Start}^C(i, j, params, P)$ query which chose i^*. If $\hat{i} = i^*$, and M's query satisfy the following constraints, then M wins; otherwise M loses.*
   - *No $\mathsf{reveal\_longterm\_keys}^P$ query for P_i and P_j.*
   - *No $\mathsf{Send}(\Psi^*, \cdot)$ query to P_i or P_j.*

A set of algorithms *AKE* is said to be a *one-way authenticated key-exchange primitive* (short 1W-AKE) if it satisfies Definitions 1, 2, and 3.

The onion algorithms. We define the properties that a triple (*FormOn*, *UnwrapOn*, *WrapOn*) of secure onion algorithms has to satisfy: onion correctness and onion security.

Recall that the additional parameter ℓ' is used for responding to a message. Since we require our construction to hide the length of the remaining path, the responder needs to give the length of the path. We call this ℓ' the appearing length of the onion.

Onion correctness states that every honestly generated forward and backward onion is correctly unwrapped and that *FormOn* can be decomposed into an algorithm *FormIn* and ℓ applications of *WrapOn*.

Definition 4 (Onion correctness)**.** *Recall that $M(\kappa)$ is the message space for the security parameter κ. Let $(P_i, k_i, sid_i)_{i=1}^{\ell}$ be a circuit, $m \in M(\kappa)$ be a message, and ℓ' be the appearing length of the onion, where $\ell \geq \ell'$. Then, we have*

- Forward correctness: *Compute $O \leftarrow FormOn((P_i, k_i, sid_i)_{i=1}^{\ell}, m, \ell')$, and for $i = 1$ to $\ell - 1$ compute $(O_{i+1}, P_i, sid_i, \ell') \leftarrow UnwrapOn(O_i, k_i)$. Then, we have $O_\ell = (m, \perp, \perp)$.*
- Backward correctness: *Compute $O_\ell \leftarrow FormOnion((P_\ell, k_\ell, sid_\ell), m, \ell')$, and for $i = \ell$ down to 2 compute $O_{i-1} \leftarrow WrapOn(O_i, P_i, k_i, sid_i)$. Thereafter, compute for $i = 1$ to $\ell - 1$ again $(O_{i+1}, P_i, sid_i, \ell') \leftarrow UnwrapOn(O_i, k_i)$. Then, we have $O_\ell = (m, \perp, \perp)$.*

Moreover, there is an algorithm FormIn such that in the two computations

$$O \leftarrow FormOn((P_i, k_i, sid_i)_{i=1}^{\ell}, m, \ell) \text{ and}$$

$$O_0 \leftarrow FormIn(\ell', m), \text{ from } i = 1 \text{ to } \ell\text{: } O_i \leftarrow WrapOn(O_{i-1}, P_i, k_i, sid_i)$$

O and O_ℓ are equally distributed. Furthermore, FormIn, FormOn, WrapOn, and UnwrapOn are efficiently computable, randomized algorithms.

The definition of *onion security* closely resembles the IND-CCA definition except for the difference that the attacker is also allowed to choose the path and the index of the challenge layer.[5] Such a strong definition is necessary for ensuring the secrecy of an onion. Consider a path in which the predecessor and the successor of the challenge node have been impersonated by the attacker. Then the attacker can easily query arbitrary onions to be wrapped or unwrapped. We stress that a *FormOn* oracle is not necessary, as *FormOn* can be decomposed into *WrapOn* and *FormIn* and *FormIn* can be computed by the attacker.

In order to hide the length of the remaining path, we require that the length of the input and the output of *UnwrapOn* are the same. The length can, e.g., be preserved by *UnwrapOn* by adding a randomly chosen padding of the right length to the onion after peeling off one layer.

Definition 5 (Onion security)**.** *Consider an adversary interacting with an onion routing challenger as follows:*

1. *Upon setup, the challenger receives from the attacker a router name P and a session id sid and generates a session key k.*
2. *The challenger answers (unwrap, O) with $a \leftarrow UnwrapOn(k, O, P)$ and (wrap, O) with $a \leftarrow WrapOn(O, P, k, sid)$.*
3. *If the adversary inputs ℓ, ℓ', a message m, two indices j, s, a path $(P_i)_{i=1}^{\ell}$ with $P_{s+1} = P$, session ids $(sid_i)_{i=1}^{\ell}$ with $sid_{s+1} = sid$, the challenger checks whether $j \in [1, \ell]$, $s \in [0, \ell]$, and $\ell \leq \ell'$, generates $\ell - 1$ session keys k_i, for $i \neq s + 1$ from the key space, draws $b \xleftarrow{\$} \{0, 1\}$, sets $k_{s+1} := k$. If $dir = \mathsf{forw}$:*
   - *If $b = 0$, let*
   $$O' \leftarrow FormOn(m, (P_i, k_i, sid_i)_{i=1}^{\ell}, \ell')$$
   For $i = 1$ to $j - 1$, compute $O_{i+1} \leftarrow UnwrapOn(O_i)$ where $O_1 := O'$.
   - *Otherwise, choose $r \xleftarrow{\$} M(\kappa) \cap \{0, 1\}^{|m|}$ and let*
   $$O_j \leftarrow FormOn(r, (P'_i, k'_i, sid'_i)_{i=j}^{s+1}), \ell')$$

   If $dir = \mathsf{back}$, send $O_j \leftarrow FormOn(x, (P_j, k_j, sid_j), \ell')$ for $x = m$ if $b = 0$ and $x = r$ for a random r as above if $b = 1$. Then, send $O_j, (k_i)_{i=j}^{s}$ to the attacker.
4. *The challenger answers for $O \neq O_s$ queries (unwrap, O) with $a \leftarrow UnwrapOn(k, O, P)$ and (wrap, O) with $a \leftarrow WrapOn(O, P, k, sid)$.*
5. *The adversary then produces a guess b'.*

We say that a set of onion algorithms satisfy onion security *if for all ppt adversaries A the probability that the attacker outputs a b' such that $b' = b$ is $1/2 + \mu(\kappa)$, where μ is negligible in κ.*

[5] Actually, it can be shown that every set of onion algorithms that satisfy onion security and correctness induces an IND-CCA secure encryption scheme (see Appendix A).

We remark that onion security implies a form of non-malleability: the attacker is not able to compute another ciphertext such that *UnwrapOn* produces an output of which the challenge message m is the same as in the challenge onion. In particular, the attacker is not able to alter the appearing length of an onion.

We say that a tuple of algorithms $(AKE, FormOn, UnwrapOn, WrapOn)$ are *secure OR modules*, if AKE is a one-way anonymous 1W-AKE (Definitions 1, 2, 3) and the onion algorithms *FormOn*, *UnwrapOn*, and *WrapOn* satisfy correctness (Definition 4) and security (Definition 5).

5 Construction and Proof

The core of an OR protocol is its key-exchange protocol and the onion forming, wrapping, and unwrapping algorithms; we call these components OR modules. We show how to construct an OR protocol out of OR modules. More precisely, we construct in Section 5.1 a template for OR protocols, and show in Section 5.2 that this template, when instantiated with secure OR modules, yields a secure OR protocol in the UC model (see Theorem 1).

5.1 Constructing an OR protocol template

We present a template for OR protocols that is parametric in the OR modules, i.e., a key-exchange protocol and onion forming, wrapping, and unwrapping algorithms.

We consider a 1W-AKE primitive since the initiator of a key exchange should remain anonymous. In this way the relation for a shared key is necessarily directed: the initiator has the role of the sender, and the responder has the role of the receiver. In order to be able to send a response to an anonymous sender over the same circuit, each router has to store its predecessor in the circuit.

We assume that every party generated an asymmetric key pair and there is a trusted (i.e., incorruptible) certification authority (CA) with whom all public keys have been registered. Moreover, we assume that this CA issues verifiable certificates for these public keys. In the UC framework such an assumption is typically realized by introducing an online key registration functionality $\mathcal{F}_{\text{REG}}$ with whom keys can be registered and from whom registered keys can be retrieved (see Section 5.1). $\mathcal{F}_{\text{REG}}$ models a CA and the security of its certificates [4].[6]

Moreover, we assume an authenticated secure channel functionality $\mathcal{F}_{\text{SCS}}$ over which all network messages are sent. In Tor, such a secure channel is realizes by a mutually authenticated TLS connection.

The protocol template Π_{OR} in detail. The onion routing protocol template Π_{OR}, illustrated in Figure 3, has an initialization phase and three main states: in the initialization phase, a party sets up its long-term keys and registers them with $\mathcal{F}_{\text{REG}}$. In the first state, the party anonymously sends a new message over a user-chosen path of the onion routing network, establishing a fresh circuit if necessary. In the second state, the router responds to the key-exchange protocol. In the third state, the router forwards an onion.

In the first state, the party P receives a **send-input** of the following form: a path $\mathcal{P}$ and a message m from the user. If no valid circuit for path $\mathcal{P}$ exists yet, the party establishes a circuit over that path using the 1W-AKE (see Section 4). A circuit $\mathcal{C}$ consists of a sequence of triples (P, k, sid): the next party P in the path, the session key k, and the corresponding session id sid. After the circuit construction, the router forms an onion by calling the procedure *FormOn* with the message m and the established circuit $\mathcal{C}$ as input.

In the second state, the party P receives a **respond-input** of the following form: a session id sid and a message m. The party P checks whether the session sid belongs to a session of which it knows the key. If the check succeeds, the party invokes the *FormOn* procedure with the length of the circuit. We stress that the length of the circuit can be computed from the length of the onion.

In the third state, the router P receives either an onion O and session id sid (in which P is the receiver) or a plaintext message. If the message contains an onion, the router checks whether it is a forward or a backward onion. For checking whether the message is a **forward onion**, the router checks whether for

[6] Technically, we consider a multi-session key registration functionality that is only used by the Π_{OR} protocol template. Therefore, only one session identifier would be used in the communication with $\mathcal{F}_{\text{REG}}$. For the sake of readability, we omit the session identifier in the message to $\mathcal{F}_{\text{REG}}$.

Upon receiving an input setup:
Generate an asymmetric key pair $(sk, pk) \leftarrow G$. Send $(\mathsf{register}, pk, P)$ to the functionality $\mathcal{F}_{\text{REG}}$. Wait until $\mathcal{F}_{\text{REG}}$ answers with a message $(\mathsf{registered}, pk)$. Then output ok.

Upon receiving an input $(\mathsf{send}, \mathcal{P}, m)$:
Do nothing if $m \notin M(\kappa)$. Otherwise, if $\mathit{established}(\mathcal{P}) = \emptyset$ and $|\mathcal{P}| < \mathit{circuit\_ttl} + 1$, call $\mathit{Extend\_Circuit}(\emptyset, \mathcal{P}, m)$. If $\mathit{established}(\mathcal{P}) = ((P_1, k_1, \mathit{sid}_1), \ldots, (P_n, k_n, \mathit{sid}_n)) = \mathcal{C}$ (for $n > 0$) and $\mathit{used}(\mathit{sid}_1) < \mathit{circuit\_ttl}$, call $\mathit{Send\_Message}(\mathcal{C}, m, P_1, \mathit{sid}_1)$. If $\mathit{used}(\mathit{sid}_1) \geq \mathit{circuit\_ttl}$, set $\mathit{established}(\mathcal{P}) := \emptyset$ and call $\mathit{Extend\_Circuit}(\emptyset, \mathcal{P}, m)$.

Upon receiving an input $(\mathsf{respond}, \mathit{sid}, m)$:
Do nothing if $m \notin M(\kappa)$. Otherwise, if $\mathit{skey}(\mathit{sid}) = (k, P_1, P)$ (for some k, P_1), let $\ell' := \mathit{circuit\_length}(\mathit{sid})$ and compute $O \leftarrow \mathit{FormOn}((P, k, \mathit{sid}), m, \ell')$ and send $(O, \mathit{sid}, P, P_1)$ over a secure channel $\mathcal{F}_{\text{SCS}}$.

Upon receiving a message $(m, \mathit{sid}, P_1, P)$ from a secure channel $\mathcal{F}_{\text{SCS}}$:

1. *(m is a forward onion)* If $m \notin \mathit{sent}$ and $\mathit{skey}(\mathit{sid}) = (k, P_1, P)$ (for some k, P_1), compute $(m', \ell') \leftarrow \mathit{UnwrapOn}(m, k)$ and distinguish the following cases.
   - (i.) If $m' = (m'', \bot)$ and $m'' = (\mathit{ake}, P_2, X, \Psi)$: Set $\mathit{pcandidate}(\Psi) := \mathit{sid}$. Send (m', P_2) over a secure channel $\mathcal{F}_{\text{SCS}}$.
   - (ii.) If $m' = (m'', \bot)$ and $m'' \neq (\mathit{ake}, P_2, X, \Psi)$: Store $\mathit{circuit\_length}(\mathit{sid}) := \ell'$, and output $(\mathsf{result}, m'', \mathit{sid})$ to party P.
   - (iii.) If $m' = (O', P_2, \mathit{sid}')$, add O' to sent and send $(O', \mathit{sid}', P, P_2)$ over a secure channel $\mathcal{F}_{\text{SCS}}$.
2. *(m is a backward onion)* We have a backward onion in the following three cases.
   - (i.) If $\mathit{predecessor}(\mathit{sid}) = \bot$, $\bot \neq ((P, (P_1, k_1, \mathit{sid}), \ldots, (P_k, k_k, \mathit{sid}_k)), P_{k+1}, \mathcal{P}, m) = \mathit{open\_AKE}(\mathit{sid})$ (in case $k = 0$ take Ψ instead of $\mathit{sid} = (\Psi, \Psi')$), increase $\mathit{used}(\mathit{sid})$, and perform the following steps:
     - (a) For $i = 1$ to k compute $(O_{i+1}, P_{i+1}, \mathit{sid}_i, l_i) \leftarrow \mathit{UnwrapOn}(O_i, k_i)$ where $O_1 := m$. If $l_{i+1} = \ell'$ proceed with the following steps.
     - (b) If the resulting message $m_{k+1} = (\mathit{ake}, P_{k+1}, t, \mathit{sid}_{k+1})$ (in case $k = 0$ $m_{k+1} = m$) and $\mathit{sid}_{k+1} = (\Psi, \Psi')$, compute $(k_{k+1}, X, Y, P_{k+1}) \leftarrow \mathit{Compute\_Key}(pk_{P_{k+1}}, m_{k+1}, \mathit{akestate}(\Psi), \Psi')$. Then, set $\mathit{akestate}(\Psi) := \bot$.
     - (c) Store $\mathit{established}(P_1, \ldots, P_{k+1}) := ((P_1, k_1, \mathit{sid}_1), \ldots, (P_{k+1}, k, \mathit{sid}_{k+1}))$ and $\mathit{skey}(\mathit{sid}_{k+1}) := (k_{k+1}, P, P_{k+1})$, and set $\mathit{open\_AKE}(\mathit{sid}) := \bot$.
     - (d) If $k + 1 < |\mathcal{P}|$, call $\mathit{Extend\_Circuit}((P_1, \ldots, P_{k+1}), \mathcal{P}, m)$; otherwise, call $\mathit{Send\_Message}(\mathcal{C}, m, P_1, \mathit{sid}_1)$.
   - (ii.) If $\mathit{predecessor}(\mathit{sid}) = \bot$, $\mathit{open\_AKE}(\mathit{sid}) = \bot$, and there is a $\mathcal{P}$ such that $\mathit{established}(\mathcal{P}) = (P_i, k_i, \mathit{sid}_i)_{i=1}^n$ and $\mathit{sid}_1 = \mathit{sid}$ (for some n), compute for $i = 1$ to k $(O_{i+1}, P_i, \mathit{sid}_i, l_{i+1}) \leftarrow \mathit{UnwrapOn}(O_i, k_i)$ where $O_1 := m$. Check whether $l_{i+1} = \ell'$, if so increase $\mathit{used}(\mathit{sid})$, and output m_n.
   - (iii.) If $\mathit{sid}' = \mathit{predecessor}(\mathit{sid})$ and $\mathit{skey}(\mathit{sid}') = (k', P_2, P)$, compute $m' \leftarrow \mathit{WrapOn}(m, (P, k', \mathit{sid}'))$ and send $(m', \mathit{sid}', P, P_2)$ over a secure channel $\mathcal{F}_{\text{SCS}}$. If m' is an onion, add m' to sent.
3. *(m is the response from a key-exchange)* If $m = (\mathit{ake}, P_1, t, \mathit{sid})$, $\mathit{sid} = (\Psi', \Psi)$ and $\mathit{sid}' = \mathit{pcandidate}(\Psi')$, set $\mathit{predecessor}(\mathit{sid}) := \mathit{sid}'$. If $\mathit{skey}(\mathit{sid}') = (k', P_2, P)$, compute $m' \leftarrow \mathit{FormOn}(P, k', \mathit{sid}', m, n)$ and send $(m', \mathit{sid}', P, P_2)$ over a secure channel $\mathcal{F}_{\text{SCS}}$.
4. *(m is the initial message from a key-exchange)* If $m = (\mathit{ake}, P, m')$ and $\mathit{sid} = \Psi$, compute $(m', (k, \star, \vec{v})) \leftarrow \mathit{Respond}(sk_P, P, m', \Psi)$. Then, let $\mathit{sid}' := (\Psi, \Psi')$, set $\mathit{skey}(\mathit{sid}') := (k, P_1, P)$, and send $(m', \mathit{sid}', P, P_1)$ over a secure channel $\mathcal{F}_{\text{SCS}}$.

Figure 3: The onion routing protocol template Π_{OR} for party P

this session the session key k is known with the router itself being the receiver. If so, the router unwraps the onion calling the procedure *UnwrapOn* with the onion O and the key k as input. We distinguish three cases for the resulting message m. In the first case, the resulting message m is an initiating key-exchange message, i.e., $m = (\mathit{ake}, x, P', y)$. Then, the router sends m over a secure channel $\mathcal{F}_{\text{SCS}}$. In the second case, the resulting message is not an onion and not a initiating key-exchange message. Then, the router outputs m to party P. In the third case, the resulting message $m = (O', P', \mathit{sid}')$ consists of an onion O' together with a party P' and a session id sid'. Then, the router sends this triple to party P' over a secure channel $\mathcal{F}_{\text{SCS}}$.

There are three cases in which a message received over $\mathcal{F}_{\text{SCS}}$ is a **backward onion**. In the first case, the party is the circuit initiator and the circuit of the onion still needs to be extended. Then, Π_{OR} confirms the circuit extension and continues extending the circuit. In the second case, the party is the circuit initiator and the circuit of the onion does not need to be extended. Then, Π_{OR} sends the message (associated with the circuit extension) over the completely established circuit. In the third case, the party is an intermediate router that forwards the onion. Π_{OR} checks whether it knows a predecessor session $sid' = predecessor(sid)$ for which it knows a key $(k, P_2, P) = skey(sid')$ (as the receiver). If the check succeeds, the router wraps the onion using *WrapOn* with the key k and sends the wrapped onion to the predecessor P_2 in the circuit.

If a **response to a key-exchange** initiation message $m = (ake, P, t, \Psi', \Psi)$ received over a secure channel, the router memorizes that the session $sid = (\Psi', \Psi)$ is in the current circuit the predecessor of sid'. Then, the router looks up for a session sid' in $(k, P', P) = skey(sid')$ the key k and the predecessor P'. Next, the router adds a layer to the response onion calling the procedures *WrapOn* with the message m and the triple (P, k, sid') as input. The resulting onion O is sent together with the session id sid' to P' over a secure channel $\mathcal{F}_{\text{SCS}}$.

If an **initial key-exchange** message $m = (ake, P, m', \Psi)$ has been received over a secure channel, the key-exchange response algorithm *Respond* is invoked, which produces a response message and a session key. The protocol template stores the session key and sends the response over a secure channel. In any other case, the router simply does nothing.

Typically, onion routing protocols, such as Tor, have a time limit for each established circuit. We model such a time limit in the UC framework by only allowing a circuit to transport at most a constant number of messages, namely *circuit_ttl* many messages. Thereafter, the circuit is discarded and a fresh circuit is established.

Subroutines of Π_{OR}. The protocol template Π_{OR} uses two subroutines: *Extend_Circuit* and *Send_Message*.

Extend_Circuit$(\mathcal{P}', \mathcal{P}, m)$: Let $(P_1, \ldots, P_\ell) := \mathcal{P}$ (for some $n \in \mathbb{N}$) and $((P_1, k_1, sid_1), \ldots, (P_s, k_s, sid_s)) := established(\mathcal{P}')$ (where $s = 0$ for $\mathcal{P}' = \emptyset$). Let $m' := (\mathsf{new\_session}, ake, P_{s+1})$, and compute $((ake, P_{s+1}, X, \Psi), state) \leftarrow Initiate(pk_{P_{s+1}}, P_{k+1}, m')$. Set $akestate(\Psi) := state$. If $k = 0$ let $sid_1 := \Psi$, store $(established(\mathcal{P}), P_{s+1}, \mathcal{P}, m)$ in $open\_AKE(sid_1)$. Then, call $Send\_Message(established(\mathcal{P}'), m', P_1, sid_1)$.

Send_Message$(\mathcal{C}, m, P', sid)$:
Let $((P_1, k_1, sid_1), \ldots, (P_\ell, k_\ell, sid_\ell)) := \mathcal{C}$.

1. If $\mathcal{C} = \emptyset$, send (m, sid, P, P') over a secure channel $\mathcal{F}_{\text{SCS}}$.
2. If $\mathcal{C} \neq \emptyset$, increase $used(sid_1)$, let $O \leftarrow FormOn(\mathcal{C}, m, \ell)$, add O to $sent$, and send (O, sid, P, P_1) over a secure channel $\mathcal{F}_{\text{SCS}}$.

Revealing the long-term keys. Upon receiving a message $(\mathsf{reveal\_longterm\_keys}, P)$ from the network: Send the long-term keys (sk, pk) over the network. Moreover, we assume that as soon as the long-term key is revealed the attacker is able to terminate the current secure channel session and establish a new secure channel session (which the attacker can now read). This is modelled by letting parties whose long-term key has been revealed forward all messages to the attacker. In other words, we let the attacker impersonate the parties.

5.2 The main proof

We say that a protocol π securely realizes $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model, if each party in the protocol π has a direct connection to $\mathcal{F}'$ (see Figure 4). Recall that $\mathcal{F}_{\text{REG}}$ is the key registration and $\mathcal{F}_{\text{SCS}}$ is the secure channel functionality. We prove our result in the $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SCS}}$-hybrid model, i.e., our result holds for any key registration and secure channel protocol securely realizing $\mathcal{F}_{\text{REG}}$, and $\mathcal{F}_{\text{SCS}}$, respectively.

Theorem 1. *If the protocol template Π_{OR} is instantiated with secure OR modules $\mathcal{M}$, then the resulting protocol $\Pi_{\text{OR}}(\mathcal{M})$ securely realizes the ideal functionality $\mathcal{F}_{\text{OR}}$ in the $\mathcal{F}_{\text{REG}}$,$\mathcal{F}_{\text{SCS}}$-hybrid model.*

Proof. We have to show that for all ppt attackers A there is a ppt simulator S such that no ppt environments E can distinguish the interaction with A and Π_{OR} from the interaction with S and $\mathcal{F}_{\text{OR}}$.

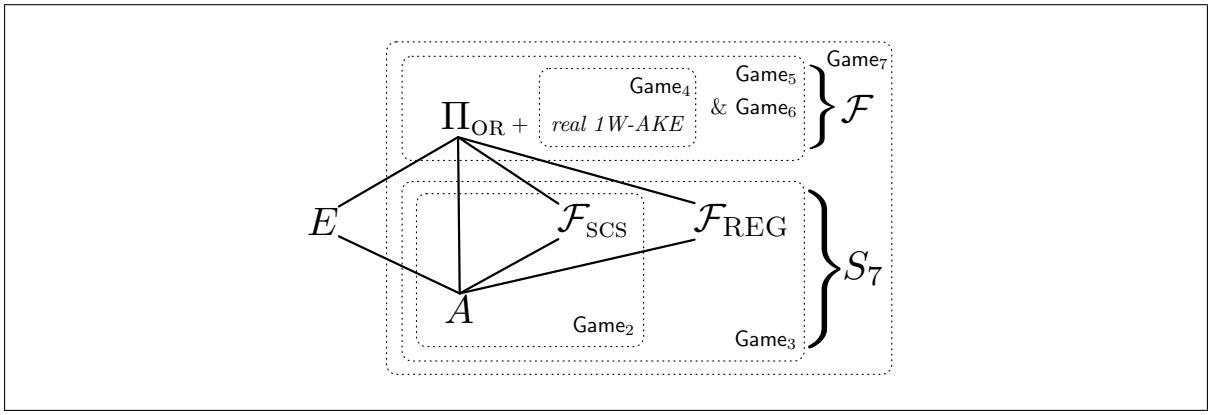

Figure 4: Proof overview - Game_1 denotes the original setting, and the dotted frames represent the changed parts in Game_i.

Given a ppt attacker A, we construct a simulator S that internally runs A and simulates the public key infrastructure, i.e, the functionality $\mathcal{F}_{\text{REG}}$. The crucial part in this proof is that the ideal functionality $\mathcal{F}_{\text{OR}}$ provides the simulator with all necessary information for the simulation. We examine a sequence of seven games, proving their pairwise indistinguishability for the environment E.

Game 1: This is the original setting in which the environment E interacts with the protocol $\Pi_1 := \Pi_{\text{OR}}(\mathcal{M})$ and the attacker A. Moreover, $\Pi_{\text{OR}}(\mathcal{M})$ and A have access to a certification authority $\mathcal{F}_{\text{REG}}$ and a secure channel functionality $\mathcal{F}_{\text{SCS}}$.

Game 2: The protocol remains the same, i.e., $\Pi_2 := \Pi_1$. The simulator S_2 computes the functionality $\mathcal{F}_{\text{SCS}}$ and internally runs the attacker A. All messages that are sent to $\mathcal{F}_{\text{SCS}}$ are captured by S_2, and all messages from and to the attacker are simply forwarded. Since S_2 honestly computes $\mathcal{F}_{\text{SCS}}$ and the attacker A, Game_1 and Game_2 are perfectly indistinguishable for the environment E.

Game 3: The protocol remains the same, i.e., $\Pi_3 := \Pi_2$. The simulator S_3 computes the functionality $\mathcal{F}_{\text{REG}}$. All messages that are sent to $\mathcal{F}_{\text{REG}}$ are captured by S_3. Since S_3 honestly computes $\mathcal{F}_{\text{REG}}$, Game_2 and Game_3 are perfectly indistinguishable for the environment E.

Game 4: We modify the session keys that have been established between two parties whose public keys have not yet been revealed. In Game_4, Π_4 maintains a shared state; in other word, all parties are one machine and share some state. Instead of using the established key, for each established key k Π_4 stores a randomly chosen value in the shared state. This random value is used as a session key instead of k. The simulator remains unchanged, i.e., $S_4 := S_3$.

Assume that there is a ppt machine that can compute a session key between two uncorrupted parties with non-negligible probability (in the security parameter κ), given the key-exchange's transcript of messages. Then, using a hybrid argument, it can be shown that there is an attacker that breaks the security of the 1W-AKE, which in turn contradicts the assumption that the OR modules are secure. Hence, Game_3 and Game_4 are computationally indistinguishable.

Game 5: In this setting the simulator remains unchanged, i.e., $S_5 = S_4$, but the protocol Π_5 uses fake onions instead of real onions. Upon receiving a send or a respond input, the protocol stores an input message m in the shared state. Then, the protocol remembers for each session whether it is malicious or not, i.e., whether one of the involved parties revealed its long-term key before the session has been established. Based on this notion of malicious session, the protocols maintains for every circuit $(P_i, k_i, sid_i)_{i=1}^{\ell}$ a separation into contiguous subsequences $(P_i, sid_i)_{i=j}^{s}$ such that sid_{j-1} and sid_{s+1} are not malicious sessions[7] or $s = \ell$ and $(sid_i)_{i=j}^{s}$ are malicious. These separations are updated upon each reveal_longterm_keys command.

Whenever in Π_4 a forward onion is sent from P to P', i.e., $P = P_{j-1}$ and $P' = P_j$, we distinguish in Π_5 two cases for $(P_i, sid_i)_{i=j}^{s}$. First, the separation is of the form $(P_i, sid_i)_{i=j}^{\ell}$, i.e., $s = \ell$. Then, the

[7] sid_0 means that the sender did not reveal its long-term key.

Upon receiving a a message $(\mathsf{reveal\_longterm\_keys}, P)$ directly from the attacker A:
Look the long-term key pair (sk_P, pk_P) of P up (using the simulated functionality $\mathcal{F}_{\text{REG}}$) and output (sk_P, pk_P). Then, forward $(\mathsf{reveal\_longterm\_keys}, P)$ to $\mathcal{F}_{\text{OR}}$.

Upon receiving a message m for the environment directly from the attacker A:
Forward m to the environment.

Upon receiving a message setup from $\mathcal{F}_{\text{OR}}$:
Generate for each party P the long-term keys and register them in $\mathcal{F}_{\text{REG}}$. Then respond ok to $\mathcal{F}_{\text{OR}}$.

Upon receiving an onion handle, i.e., a message $m = (h, l_c, l_m, P_1, Q, sid, P_2)$ or $m = (h, l_c, l_m, P_1, Q, P_2)$ from $\mathcal{F}_{\text{OR}}$:
Let $Q = (P_i, sid_i)_{i=j}^{s}$. Construct an onion O for the path $(P_i, skey(sid_i), sid_i)_{i=j}^{s}$ containing a randomly chosen string of length l_m. For honest sessions sid_u no key exchange is performed and $skey(sid_u)$ constitutes a randomly chosen key. For malicious sessions sid_u, the established key is stored in $skey(sid_u)$. If $skey(sid_i)$ is undefined, i.e., sid_i is a malicious session and the key has not yet been established, stop. Store the handle h as $pending(h) := (O, (P_s, sid_s))$, and send (O, P_1, sid_j, P_j) over $\mathcal{F}_{\text{SCS}}$ to P_j.

Upon receiving an output to an attacker-impersonated party P, i.e., a message (m, Q) from $\mathcal{F}_{\text{OR}}$:
Let $Q = (P_i, sid_i)_{i=j}^{s}$. Construct an onion for the path $(P_i, skey(sid_i), sid_i)_{i=1}^{s}$ containing m, where $skey$ is defined as above. If $skey(sid_i)$ is not known, stop.

Upon receiving an onion (m, sid, P_1, P) from $\mathcal{F}_{\text{SCS}}$ for party P:
Act as Π. If P is the receiver of a backward onion that carries a response to a key establishment, run *Compute_Key*, yielding a session id sid' and a key k. Then, set $skey(sid') := k$. Recall that the simulator also checks after a *UnwrapOn* call whether the resulting circuit length still equals the stored appearing length $l_c = \ell'$ (sent by $\mathcal{F}_{\text{OR}}$).

If Π would output a message m to a party P, send $(\mathsf{attacker\_message}, m, P)$ to $\mathcal{F}_{\text{OR}}$. Otherwise, look for an h such that $pending(h) = (O, (P, sid))$ and send $(\mathsf{deliver}, h, P_1, P)$ to $\mathcal{F}_{\text{OR}}$.

Upon receiving the initial message from a key-exchange (m, sid, P_1, P) from $\mathcal{F}_{\text{SCS}}$ for party P:
Run *Respond* as in Π obtaining a session id sid' and a response m', and store the key in $skey(sid')$. Then, construct an onion response m'.

Figure 5: The final simulator $S = S_7$ that internally runs the attacker A

protocol constructs an onion using keys $k_j, \ldots, k_\ell$ and the real message, which is stored in the shared state. Recall that, as in Game_4, an honest session key is chosen at random and read from the shared state.Second, the separation is of the form $(P_i, sid_i)_{i=j}^{s}$ and sid_{s+1} is not malicious and $s+1 \leq \ell$. Then, the protocol constructs an onion using randomly chosen keys $k_j, \ldots, k_{s+1}$ and a randomly chosen message of the same length as the real message.

Whenever in Π_4 a backward onion is sent from P to P', i.e., $P = P_{s+1}$ and $P' = P_s$, we again distinguish in Π_5 two cases. First, the separation is of the form $(P_i, sid_i)_{i=1}^{s}$, i.e., the sessions with all remaining parties back to the initiator are malicious, i.e., $j = 1$. Then, we form the layer with (P_i, k_i, sid_i) and the real message m, which we read from the shared state. Second, the separation is of the form $(P_i, sid_i)_{i=j}^{i}$ and sid_{j-1} is honest, i.e., there is at least one session on the remaining way in the circuit that is not malicious. Then, we form an onion with (P_i, k_i, sid_i) and a random value r of the same length as m.

Every randomly chosen string that is used as instead of a real message m in the construction of an onion, is connected to exactly one message m. In the communication between two honest parties, this m (stored in the shared state) is output instead of the random string that is contained in the onion. In particular, if m is a key-establishment message Π_5 does not output m but answers according to the key establishment message, again replacing m with a random string if the circuit does not contain any honest session.

We stress that Π_5 remembers the appearing length ℓ' of an onion and checks upon each *UnwrapOn* execution whether the computed circuit length equals ℓ'. If the check fails, the respective party stops (and the control is handed over to the environment).

By a hybrid argument it follows that any attacker distinguishing Game_5 from Game_4 can be used for breaking onion security (Definition 5), which contradicts the assumed security of the OR modules. Hence, Game_5 and Game_4 are indistinguishable. The ith hybrid is constructed by faking all messages from the ith honest session on, where i is a global enumeration of all sessions by honest parties.

Game 6: In this setting the simulator remains unchanged, i.e., $S_6 = S_5$, but the protocol Π_6 in addition internally runs the ideal functionality $\mathcal{F}_{\text{OR}}$. We stress that every network message $(h, \mathit{length}, P_j, \mathcal{C}')$ of the ideal functionality corresponds to a separation $(P_i, \mathit{sid}_i)_{i=j}^{s}$ of some circuit: (i) either $\mathcal{C}' = ((P_i, \mathit{sid}_i)_{i=j}^{s}, P_{s+1})$ or (ii) $\mathcal{P}' = ((P_i, \mathit{sid}_i)_{i=j}^{\ell}, m)$ and $s = \ell$. Hence, we let Π_6 compute the onion based on the network message $(h, \mathit{length}, \mathcal{P}')$. As in Game_5, in case (i) Π_6 computes a fake onion, and in case (ii) Π_6 computes a real onion using m beginning from P_j.

As Π_6 uses random keys, it can as well fake the circuit establishment messages except for attacker-impersonated receivers. But in that case, as in Game_5, Π_6 sends the real message, since the functionality provides Π_6 with all necessary information. In all other cases, Π_6 behaves just like Π_5, because the ideal functionality outputs only then the real messages and session ids when also Π_5 would need them.

Game 7: In this setting we replace the protocol with the ideal functionality, i.e., $\Pi_7 := \mathcal{F}_{\text{OR}}$. The simulator S_7 in Game_7 additionally computes all network messages exactly as Π_6.

The ideal functionality behaves towards the environment exactly as Π_{OR}; consequently, it suffices to show that the network messages are indistinguishable. However, as already Π_6 complied with the ideal functionality concerning the network messages and the simulator is just defined like Π_6 concerning the network messages, Game_6 and Game_7 are indistinguishable. □

We stress that anonymity in a low-latency OR network does not solely depend upon the cryptographic security. It is rather mostly obtained from factors such as magnitudes and distributions of users and their destination servers. In the OR literature, considerable effort has been put towards measuring the anonymity of onion routing [8, 9, 19, 23], regarding the OR network as a black-box. We emphasize that our UC-secure OR construction provides such a black-box security guarantee.

6 An Instantiation of Secure OR Modules

In this section, we construct secure OR modules $\mathcal{M}$ as defined in Section 4 and conclude, by Theorem 1, that $\Pi_{\text{OR}}(\mathcal{M})$ constitutes a provably secure and practical OR protocol.

The one-way anonymous 1W-AKE. We use the ntor protocol, which has been proposed by Goldberg, Stebila, and Ustaoglu [11], and proven to constitute a 1W-AKE. The ntor protocol is presented in Figure 6. Although their result uses the random oracle model (ROM), random oracles are not an intrinsic requirement for constructing 1W-AKEs. We decided to use ntor since, to the best of our knowledge, it currently constitutes the most efficient 1W-AKE.

Lemma 1 (ntor is anonymous and secure [11])**.** *The* ntor *protocol is a one-way anonymous and secure one-way authenticated key-exchange protocol in the ROM.*

Construction 1: The onion algorithms. Based on the construction from Camenisch and Lysyanskaya [3], we propose in Figure 7 a construction for the onion algorithms *FormOn*, *UnwrapOn*, and *WrapOn*. Let (G_p, D_p, E_p) be a PRP, (G_e, D_e, E_e) an IND-CCA symmetric encryption scheme, and let $\mathcal{H}$ be a collision-resistant hash function. We require that the message space of the PRP and the IND-CCA scheme coincide with $M(\kappa)$. We write $\{m\}_k := E_p(m, k)$ and $\{c\}_k^{-1} := D_p(c, k)$.

Lemma 2. *FormOn_I and $\mathit{UnwrapOn}_I$, and WrapOn_I from Construction 1 satisfy correctness and security for onion algorithms.*

Proof. The correctness follows by construction. Hence, it remains to show the security. This is shown in a sequence of games. Game_1 is simply the setting from Definition 5 if $b = 0$.

Upon $Initiate(pk_Q, Q, \mathsf{new\_session}, \mathsf{ntor})$:

1. Generate the ephemeral key pair (x, g^x).
2. Set session id $\Psi_P = H_{sid}(g^x)$.
3. Set $state(\Psi_P) := (\mathsf{ntor}, Q, x, g^x)$.
4. Set $m := (\mathsf{ntor}, \Psi_P, Q, g^x)$.
5. Output $(m, state(\Psi_P))$.

Upon $Respond(sk_P, P, X, \Psi)$:

1. Verify that $X \in G^*$.
2. Generate the ephemeral key pair (x, g^x).
3. Set session id $\Psi_P = H_{sid}(g^x)$.
4. Compute $(sk', sk) = H(X^x, X^{sk_P}, P, X, g^x, \mathsf{ntor})$.
5. Compute $t_P = H_{mac}(sk', P, g^x, X, \mathsf{ntor}, \mathsf{server})$.
6. Set $m := (\mathsf{ntor}, g^x, t_P, \Psi_P, \Psi)$.
7. Set $out := (sk, \star, X, g^x, P)$.
8. Output (m, out).

Upon $Compute\_Key(pk_Q, Y, t_Q, \Psi', \Psi)$:

1. Check whether $state(\Psi') = (\mathsf{ntor}, Q, x, g^x)$ (for some Q and x). If the check fails, abort and output $\perp$.
2. Verify that $Y \in G^*$.
3. Compute $(k', k) = H(Y^x, pk_Q^x, Q, g^x, Y, \mathsf{ntor})$.
4. Verify $t_B = H_{mac}(k', Q, Y, g^x, \mathsf{ntor}, \mathsf{server})$.
5. Output (k, Q, g^x, Y, Q).

If any verification fails, the party erases all session-specific information and aborts the session.

Figure 6: The ntor protocol

In Game_2, $FormOn_I$ is only applied to the circuit $(P_i, k_i, sid_i)_{i=j}^{\ell}$ and the message m. By the correctness of the onion algorithms, Game_1 and Game_2 are perfectly indistinguishable.

In Game_3, $FormOn_I$ uses, from layer s on, instead of the PRP E_p and D_p a randomly chosen function, which encrypts a randomly chosen string of the same length as the input. Similarly $WrapOn_I$ uses for all queries the randomly chosen function. By the pseudorandomness of (G_p, E_p, D_p), Game_2 and Game_3 are computationally indistinguishable for the attacker, as it does not know the encryption key k_s^p.

In Game_4, $FormOn_I$ and $WrapOn_I$ still use the PRP; however, they additionally now also use the *Fake_Encrypt* and *Fake_Decrypt* algorithm (from the IND-CCA game[8]) instead of the IND-CCA secure encryption scheme E_e. By the IND-CCA property Game_3 and Game_4 are indistinguishable, since the attacker does not know the key k_s^e.

In Game_5, $FormOn_I$ and $WrapOn_I$, which still use the randomly chosen function and *Fake_Encrypt* from layer ℓ on, is applied to $(r, (P_i', k_i', sid_i')_{i=j}^{s+1}, \ell')$ instead of $(m, (P_i, k_i, sid_i)_{i=1}^{\ell}, \ell')$, where j and s are chosen by the attacker as in the onion security game. We stress that $(P_i', k_i', sid_i') = (P_i, k_i, sid_i)$ for $i = 1$ to s. Moreover in *Fake_Encrypt* the message is not used at all, and the output of the randomly chosen function is uniformly distributed, Game_3 and Game_4 are perfectly indistinguishable.

In Game_6, $FormOn_I$ and $WrapOn_I$ use the PRP E_p and D_p and the IND-CCA encryption scheme E_e and D_e again instead of the randomly chosen function and *Fake_Encrypt*. By the pseudorandomness of E_p and D_p and the IND-CCA property of E_e and D_e the Game_5 and Game_6 are computationally indistinguishable. As we can see, Game_6 equals the onion security challenger in the case $b = 1$. Hence, the two cases $b = 0$ and $b = 1$ are computationally indistinguishable for this construction. □

As mentioned earlier, we only need the ROM because the security proof of ntor uses it [11]. Any other protocol that can be proven to be a 1W-AKE without a RO yields a protocol that can be proven secure without ROs.

[8] Actually, it is from the ROR-CCA (real-or-random) game. But Bellare, Desai, Jokipii, and Rogaway showed that IND-CCA and ROR-CCA are poly-time equivalent [2].

$FormOn_I(m, (P_i, k_i, sid_i)_{i=1}^{\ell}, \ell')$:

1. *Initialization.* If $\ell \leq \ell'$, let $P_i := P_\ell$ for $\ell < i \leq \ell'$ and $k_{\ell+1} := \cdots := k_{\ell'} := k_\ell$. Let $k_i^p := G_p(k_i)$ and $k_i^e := G_e(k_i)$, where $G_p(k_i)$ denotes that G_p uses k_i as randomness, and analogously for G_e.
2. *Form inner layer.* Let $R_i := \{P_i, sid_i\}_{k_i^p}^{-1}$ and $M^{(\ell')} := \{m\}_{k_{\ell'}^p}$. Let $H_i^{\ell'} := \{\ldots\{R_i\}_{k_{i+1}^p}^{-1}\ldots\}_{k_{\ell'-1}^p}^{-1}$ for $i \in [1, \ell'-1]$; in particular, we have $H_{\ell'-1}^{\ell'} := R_{\ell'-1}$. Let $T_{\ell'} := \mathcal{H}(M^{(\ell')}, H_{\ell'-1}^{(\ell')}, \ldots, H_1^{(\ell')})$. Finally, let $C_{\ell'} \leftarrow E_e(k_{\ell'}, T_{\ell'})$. Let $O_{\ell'} := (M^{(\ell')}, H_{\ell'-1}^{(\ell')}, \ldots, H_1^{(\ell')}, C_{\ell'})$.
3. *Adding a layer.* Given O_i (for $i \in [2, \ell']$), we compute O_{i-1} as follows: $M^{(i-1)} := \{M^{(i)}\}_{k_{i-1}^p}$, $H_j^{(i-1)} := \{H_{j-1}^{(i)}\}_{k_{i-1}}$ for $j \in [2, \ell']$, and $H_1^{(i-1)} := \{P_i, sid_i, C_i\}_{k_{i-1}^p}$. Let $T_{i-1} = \mathcal{H}(M^{(i-1)}, H_{N-1}^{(i-1)}, \ldots, H_1^{(i-1)})$. Finally, let $C_{i-1} := D_e(k_{i-1}^e, T_{i-1})$. The resulting onion is $O_{i-1} = (M^{(i-1)}, H_{\ell'}^{(i-1)}, \ldots, H_1^{(i-1)}, C_{i-1})$.

$UnwrapOn_I((M, H_{\ell'}, \ldots, H_1, C), k)$:

1. Let $k^p := G_p(k)$ and $k^e := G_e(k)$.
2. Let $T := \mathcal{H}(M, H_{\ell'}, \ldots, H_1)$ and check $T = D_e(k^e, C)$.
3. Compute $(P', sid', C') := \{H_1\}_{k^p}^{-1}$.
4. Choose at random a string r of length $|\{H_{\ell'}\}_k|$.
5. Output $((((\{M\}_{k^p}^{-1}, \{r\}_{k^p}^{-1}, (\{H_{\ell'}\}_{k^p}^{-1})_{i=\ell'}^{2}, C'), P', sid'), \ell')$ (or $(m, \perp, bot)$ in the last layer).

$WrapOn_I((M, H_{\ell'}, \ldots, H_1, C), P, k, sid)$:

1. Let $k^p := G_p(k)$ and $k^e := G_e(k)$.
2. Compute $M' := \{M\}_{k^p}$, $H_j' := \{H_{j-1}\}_{k^p}$ for $1 < j \leq \ell'$, and $H_1' = \{P, sid, C\}_{k^p}$.
3. Let $T' = \mathcal{H}(M, H_{\ell'-1}, \ldots, H_1)$ and $C' \leftarrow E_e(k^e, T)$.
4. Output $(M', (H_i')_{i=\ell'}^{1}, C')$.

If any of these operations fails, output $\perp$.

Figure 7: Construction 1: The onion algorithms

Theorem 2. *Let* ntor_scheme *denote the OR modules* (ntor, $FormOn_I$, $UnwrapOn_I$, $WrapOn_I$) *from Figure 6 and Construction 1. Then, in the ROM the protocol* Π_{OR}(ntor_scheme) *using securely realizes the ideal functionality* $\mathcal{F}_{\text{OR}}$ *in the* $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SCS}}$*-hybrid model.*

Proof. This result directly follows from Lemma 1, Lemma 2, and Theorem 1. □

Goldberg and Danezis [6] present in Sphinx an alternative implementation of the onion algorithms that also satisfies the definition of Camenisch and Lysyanskaya [3]. Their construction can also be easily adjusted to satisfy our definition, yielding a more efficient construction. Because of space constraints, however, we do not present the Sphinx construction here.

7 Conclusion and Future Work

We have presented a security definition in the universal composability framework of the multi-pass onion routing (OR) circuit construction, which is also the approach that is deployed in the existing Tor protocol. Moreover, we have presented security definitions for the core modules used in an OR protocol: a one-way authenticated key exchange (1W-AKE) primitive, and onion forming, wrapping and unwrapping algorithms. Then, we have shown that secure OR modules induce a canonical OR protocol that satisfies our definition. Finally, we have instantiated our definition with a recently introduced efficient and provably secure key exchange protocol, thereby establishing the first security proof for a practical OR protocol with multi-pass circuit construction, such as Tor.

Although our work proposes a provably secure, efficient replacement for the existing OR construction Tor, users' anonymity may still be adversely affected if different users use different versions of the OR protocol. Hence it is an important direction for future work to develop a anonymity-preserving methodology for updating OR clients.

Acknowledgements. This work is partially supported by NSERC and MITACS. We thank Gregory Zaverucha for helpful preliminary discussions, and Paul Syverson for his valuable comments on an earlier draft of the paper.

References

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

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 1997.

[3] J. Camenisch and A. Lysyanskaya. A formal treatment of onion routing. In *Advances in Cryptology—CRYPTO 2005*, pages 169–187, 2005.

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

[5] D. Catalano, D. Fiore, and R. Gennaro. Certificateless onion routing. In *Proc. 16th ACM Conference on Computer and Communication Security (CCS)*, pages 151–160, 2009.

[6] G. Danezis and I. Goldberg. Sphinx: A Compact and Provably Secure Mix Format. In *Proc. 30th IEEE Symposium on Security & Privacy*, pages 269–282, 2009.

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

[8] J. Feigenbaum, A. Johnson, and P. F. Syverson. A model of onion routing with provable anonymity. In *Proc. 11th Conference on Financial Cryptography and Data Security (FC)*, pages 57–71, 2007.

[9] J. Feigenbaum, A. Johnson, and P. F. Syverson. Probabilistic analysis of onion routing in a black-box model. In *Proc. 6th ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 1–10, 2007.

[10] I. Goldberg. On the Security of the Tor Authentication Protocol. In *Proc. 6th Workshop on Privacy Enhancing Technologies*, pages 316–331, 2006.

[11] I. Goldberg, D. Stebila, and B. Ustaoglu. Anonymity and one-way authentication in key exchange protocols. Technical Report CACR 2011-11, University of Waterloo Centre for Applied Cryptographic Research, 2011.

[12] D. M. Goldschlag, M. Reed, and P. Syverson. Hiding Routing Information. In *Proc. 1st Workshop on Information Hiding*, pages 137–150, 1996.

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

[14] A. Greenberg. How To Protect Yourself Online Like An Arab Revolutionary. `http://blogs.forbes.com/andygreenberg/2011/03/25/`, 2011.

[15] A. Kate and I. Goldberg. Distributed Private-Key Generators for Identity-Based Cryptography. In *Proc. 7th Conference on Security and Cryptography for Networks (SCN)*, pages 436–453, 2010.

[16] A. Kate and I. Goldberg. Using Sphinx to Improve Onion Routing Circuit Construction. In *Proc. 14th Conference on Financial Cryptography and Data Security (FC)*, pages 359–366, 2010.

[17] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing. In *Proc. 7th Privacy Enhancing Technologies Symposium (PETS)*, pages 95–112, 2007.

[18] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-Based Onion Routing with Improved Forward Secrecy. *ACM Trans. Inf. Syst. Secur.*, 13(4):29, 2010.

[19] S. Mauw, J. Verschuren, and E. P. de Vink. A Formalization of Anonymity and Onion Routing. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, pages 109–124, 2004.

[20] L. Øverlier and P. Syverson. Improving efficiency and simplicity of tor circuit establishment and hidden services. In *Proc. 7th Privacy Enhancing Technologies Symposium (PETS)*, pages 134–152, 2007.

[21] T. T. Project. . `https://www.torproject.org/`, 2003. Accessed May 2011.

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

[23] V. Shmatikov. Probabilistic analysis of an anonymity system. *Journal of Computer Security*, 12(3-4):355–377, 2004.

[24] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In *Proc. Workshop on Design Issues in Anonymity and Unobservability (WDIAU)*, pages 96–114, 2000.

Appendix

A Onion Security implies IND-CCA

Given a set of secure onion algorithms (*FormOn*, *UnwrapOn*, *WrapOn*), we construct an IND-CCA secure symmetric key encryption (G, E, D)scheme as follows:

- G is defined as follows: choose at random a bitstring k from the key space of the onion algorithms. Then, output k.
- $E(k, m)$: Run $O \leftarrow \mathit{FormOn}(m, (0^{q_1}, k, 0^{q_2}), 1)$, where q_1 is the corresponding length of the party id and q_2 of the session id. Then, output O.
- $D(k, c)$: Run $m \leftarrow \mathit{UnwrapOn}(c, k)$. Output m.

The decryption routine is deterministic, since the algorithm *UnwrapOn* is deterministic when applied to the last layer. We prove the IND-CCA security by showing that any ppt attacker that is able to break the IND-CCA property of (G, E, D) can be used to break the onion secrecy of the underlying onion algorithms (*FormOn*, *UnwrapOn*, *WrapOn*).

Given a ppt machine A, i.e., an adversary, against the IND-CCA challenger, we construct an adversary S against the onion security game such that the success probability of A being non-negligible implies the success probability of S being non-negligible.

Construction of S. S constitutes both the IND-CCA challenger and the adversary for the onion security challenger. Hence, S has to implement the encryption oracle $\mathcal{E}_b(k, \cdot)$ and the description oracle $\mathcal{D}_d(k, \cdot)$ (for a key k). S is constructed as follows:

- Initially, S sends to the onion security challenger a pair $(0^{q_1}, 0^{q_2})$.
- Upon a request c from A to the decryption oracle, S sends a request (unwrap, c) to the $\mathit{UnwrapOn}(\cdot, 0^{q_1}, k)$ oracle. The response m is forwarded to A.
- Upon a request m from A to the encryption oracle, S computes $O \leftarrow \mathit{FormIn}(1, m)$ sends a request (wrap, O) to the $\mathit{WrapOn}(\cdot, 0^{q_1}, k, 0^{q_2})$ oracle. The response c is forwarded to A.
- Upon two challenge message m_1, m_2 (such that $|m_1| = |m_2|$) from the attacker, S chooses a random bit b sends the message m_b, the path 0^{q_1}, the session id 0^{q_2}, the indices $j := 1$ and $s := 0$, and pathlenghts $\ell := 1$ and $\ell' := 1$ to the onion security challenger. The response O is forwarded to A.
- Upon receiving the final decision bit b^* of A, S checks whether $b^* = b$. If the check succeeds, S outputs b^*; otherwise, S outputs a random number.

Let b_o be the bit that the onion security challenger flipped. If $b_o = 1$, i.e., the message has been replaced by a random string, S guessed the correct coin with probability $1/2$. If b_o, i.e., the actual message m_b has been used, and if $b^* = b$, then S has the same advantage as A. Since this $b_o = 0$ with probability $1/2$, the advantage of S is greater or equal to than $1/2$ times the advantage of A. Hence, if A breaks the IND-CCA property of (G, E, D) with non-negligible probability, than S breaks the onion security of (*FormOn*, *WrapOn*, *UnwrapOn*) with non-negligible probability.