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 appears satisfactory, a comprehensive analysis of its security guarantees is still lacking. This has also resulted in a significant gap between research work on OR protocols and existing OR anonymity analyses. In this work, we address both issues with onion routing by defining a provably secure OR protocol, which is practical for deployment in the next generation Tor network.

We start off by presenting a security definition (an ideal functionality) for the OR methodology in the universal composability (UC) framework. We then determine the exact security properties required for OR cryptographic primitives (onion construction and processing algorithms, and a key exchange protocol) to achieve a provably secure OR protocol. We show that the currently deployed onion algorithms with slightly strengthened integrity properties can be used in a provably secure OR construction. In the process, we identify the concept of predictably malleable symmetric encryptions, which might be of independent interest. On the other hand, we find the currently deployed key exchange protocol to be inefficient and difficult to analyze and instead show that a recent, significantly more efficient, key exchange protocol can be used in a provably secure OR construction.

In addition, our definition greatly simplifies the process of analyzing OR anonymity metrics. We define and prove forward secrecy for the OR protocol, and realize our (white-box) OR definition from an OR black-box model assumed in a recent anonymity analysis. This realization not only makes the analysis formally applicable to the OR protocol but also identifies the exact adversary and network assumptions made by the black box model.

Contents

1 Introduction **3**
1.1 Our Contributions 3

2 Background **4**
2.1 Onion Routing Circuit Construction 4
2.2 One-Way Authenticated Key Exchange—1W-AKE 5
2.3 The OR Protocol 6
2.4 The UC Framework: An Overview 8
2.5 An OR Black Box Model 9

3 Security Definition of OR **10**
3.1 System and Adversary Model 10
3.2 Ideal Functionality 10

4 Secure OR modules **13**
4.1 Predictably Malleable Encryption 14
4.2 Secure Onion Algorithms 15
4.3 One-Way Authenticated Key-Exchange 17

5 Π_{OR} UC-Realizes $\mathcal{F}_{\text{OR}}$ **19**

6 Instantiating Secure OR Modules **21**
6.1 Deterministic Counter Mode and Predictable Malleability 22
6.2 Security of Tor's Onion Algorithms 22
6.3 ntor: A 1W-AKE 24

7 Forward Secrecy and Anonymity Analyses **25**
7.1 OR Anonymity Analysis 25
7.2 Forward Secrecy 29

8 Conclusions and Future Work **32**

1 Introduction

Over the last few years the onion routing (OR) network Tor [32] has emerged as a successful technology for anonymous web browsing. It currently employs more than two thousand dedicated relays, and serves hundreds of thousands of users across the world. Its impact is also evident from the media coverage it has received over the last few years [17]. Despite its success, the existing Tor network still lacks a rigorous security analysis, as its circuit construction as well as network transmission delays are found to be large [26,28], the current infrastructure is not scalable enough for the future users [24,25,27], and from the cryptographic point of view its security properties have neither been formalized cryptographically nor proven. (See [3, 10, 23] for previous attempts and their shortcomings.) In this paper, we define security for the third-generation 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 routing information for the OR nodes to the users. A user (say Alice) constructs a *circuit* by choosing a small sequence of (usually three) OR nodes, where the chosen nodes route Alice's traffic over the path formed. The crucial property of an OR protocol is that a node in a circuit can determine no circuit nodes other than its predecessor and its successor. Alice sends data over the constructed circuit 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), cryptographic challenges in onion routing are to securely agree upon such symmetric keys, and then to use the symmetric keys to achieve confidentiality and integrity.

In the first generation onion routing [29], circuits are constructed in a single pass. However, the scalability issues while pursuing forward secrecy [6] in the single-pass construction prompted Dingledine, Mathewson and Syverson [8] to use a telescoping approach for the next-generation OR protocol Tor. In this telescoping approach, they employed a forward secret, *multi-pass* key agreement protocol called the Tor authentication protocol (TAP) to negotiate a symmetric session key between user Alice 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 [12] presented a security proof for individual instances of TAP. The security of TAP, however, does not automatically imply the security of the Tor protocol. (For a possible concurrent execution attack, see [33]). The Tor protocol constitutes a sequential execution of multiple TAP instances as well as onion construction and processing algorithms, and thus its security 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, and presented a protocol construction that satisfies their definition. They motivated their choice of the UC framework for a security definition by its versatility as well as its appropriateness for capturing protocol compositions. However, Feigenbaum, Johnson and Syverson [10, 11] observe that the protocol definition presented by Camenisch and Lysyanskaya [3] does not correspond to the OR methodology, and a rigorous security analysis of an OR protocol still remains an unsolved problem.

Studies on OR anonymity such as [10, 23, 30] assume simplified OR black-box models to perform an analysis of the anonymity guarantees of these models. Due to the complexity of an OR network's interaction with the network and the adversary, such black-box models are not trivially realized by deployed OR networks, such as Tor. As a result, there is a gap between deployed OR protocols and anonymity analysis research that has to be filled.

1.1 Our Contributions

Our contribution is threefold. First, we present a security definition for the OR methodology as an ideal functionality $\mathcal{F}_{\text{OR}}$ in the UC framework. This ideal functionality in particular gives appropriate considerations to the goals of various system entities. After that, we identify and characterize which cryptographic primitives constitute central building blocks of onion routing, and we give corresponding security definitions: a one-way authenticated key exchange (1W-AKE) primitive, and onion construction and processing algorithms. We then describe an OR protocol Π_{OR} that follows the current Tor specification and that relies on these building blocks as black boxes. We finally show that Π_{OR} is secure in the UC framework with respect to $\mathcal{F}_{\text{OR}}$, provided that these building blocks are instantiated with secure realizations (according to their respective security definitions).

Second, we present a practical OR protocol by instantiating Π_{OR} with the following OR modules:

a 1W-AKE protocol ntor [13], employed onion construction and processing algorithms in Tor with a slightly enhanced integrity mechanism. We show that these instantiations fulfill the security definitions of the individual building blocks that we identified before. This yields the first practical and provably secure OR protocol that follows the Tor specification. As part of these proofs, we identify a novel security definition of symmetric encryption notion we show to be sufficient for showing Π_{OR} secure. This notion strictly lies between CPA-security and CCA-security and characterizes stateful deterministic countermode encryptions. We call this notion *predictably malleable encryptions*, which might be of an independent interest.

Third, we illustrate the applicability of the abstraction $\mathcal{F}_{\text{OR}}$ by introducing the first cryptographic definition of forward circuit secrecy for onion routing, which might be of independent interest. We utilize the abstraction $\mathcal{F}_{\text{OR}}$ and the UC composability theorem for proving that Π_{OR} satisfies forward circuit secrecy by means of a simple proof. As a second application, we close the gap between the OR black-box model, prevalently used in anonymity analyses [10,11,23,30], and a cryptographic model (Π_{OR}) of onion routing. Again, we utilize our abstraction $\mathcal{F}_{\text{OR}}$ and the UC composability theorem for proving that against local, static attackers the recent analysis of the OR black-box model [11] also applies to our OR protocol Π_{OR} instantiated with secure core building blocks.

Compared to previous work [3], we construct an OR circuit interactively in multiple passes, whereas previous work did not consider circuit construction at all, and hence does not model the widely used Tor protocol. The previous approach, and even single-pass circuit construction in general, restricts the protocol to eventual forward secrecy, while a multi-pass circuit construction ensures forward secrecy immediately after the circuit is closed. Second, we show that their hop-to-hop integrity verification is not mandatory, and that an end-to-end integrity verification suffices for onion routing. Finally, they do not consider backward messages (from web-servers to Alice), and their onion wrapping and unwrapping algorithms also do not work in the backward direction.

Another important approach for analyzing onion routing has been conducted by Feigenbaum, Johnson, and Syverson [9]. In contrast to our work, the authors analyze an I/O automaton that use idealized encryption, pre-shared keys, and assume that every party only constructs one circuit to one destination. Moreover, the result in that work only holds in the stand-alone model against a local attackers whereas our result holds in the UC model against global and partially global attackers. In particular, by the UC composability theorem our result even holds with arbitrary protocols surrounding and against an attacker that controls parts of the network.

Outline of the Paper. Section 2 provides background information relevant to onion routing, 1W-AKE, and the UC framework. In Section 3, we present our security definition for onion routing. In Section 4, we present cryptographic definitions for the 1W-AKE primitive and onion construction and processing algorithms. In Section 5, we prove that given a set of secure OR modules we can construct a secure OR protocol. In Section 7, we use our ideal functionality to analyze some security and anonymity properties of onion routing. Finally, we conclude and discuss some further interesting directions in Section 8.

2 Background

In this paper, we often omit the security parameter κ when calling an algorithm A; i.e., we abbreviate $A(1^\kappa, x)$ by $A(x)$. We write $y \leftarrow A(x)$ for the assignment of the result of $A(x)$ to a variable y, and we write $y \stackrel{\$}{\leftarrow} S$ for the assignment of a uniformly chosen element from S to y. For a given security parameter κ, we assume a message space $M(\kappa)$ that is disjoint from the set of onions. We assume a distinguished error message $\perp$; in particular, $\perp$ is not in the message space. For some algorithms, we write $Alg(a, b, c, [d])$ and mean that the argument d is optional. Finally, for stateful algorithms, we write $y \leftarrow A(x)$ but we actually mean $(y, s') \leftarrow A(x, s)$, where s' is used in the next invocation of A as a state, and s is the stored state from the previous invocation.We assume that for all algorithms $s \in \{0, 1^\kappa\}$. We abbreviate probabilistic polynomial-time as PPT.

2.1 Onion Routing Circuit Construction

In the original Onion Routing project [15,16,29,31], circuits were constructed in a single pass. However, such a single-pass circuit construction does not provide *forward secrecy*: if an adversary corrupts a node and obtains the private key, 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 the start of each interval every user has to download a new set of public keys for all the nodes.

A user (Alice) 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 Alice. 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.

There are attempts to solve this scalability issue. Kate, Zaverucha and Goldberg [21] 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 a certificateless cryptography (CLC) setting [1] instead, and defined two certificateless onion routing protocols (CL-OR and 2-CL-OR). However, both approaches do not yield satisfactory solutions: CL-OR and 2-CL-OR suffer from the same scalability issues as the original OR protocol [20]; PB-OR requires a distributed private-key generator [19] that may lead to inefficiency in practice.

Another problem with the single-pass approach is its intrinsic restriction to *eventual* forward secrecy [22].; i.e., if the current private key is leaked, then past sessions remain secret only if their public and private keys have expired. 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.

In the current Tor protocol, circuits are constructed using a multi-pass approach that is based on TAP. 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 knowing the private key does not suffice for reconstructing the session key. These temporary values are erased immediately after the session key has been computed. This technique achieves immediate forward secrecy in multi-pass constructions, which however was never formally defined or proven before.

The multi-pass approach incurs additional communication overhead. However, in practice, almost all Tor circuits are constructed for a circuit length of $\ell = 3$, which merely causes an overhead of six additional messages.[1] With this small overhead, the multi-pass circuit construction is the preferred choice in practice, due to its improved forward secrecy guarantees. Consequently, for our OR security definition we consider a multi-pass circuit construction as in Tor.

2.2 One-Way Authenticated Key Exchange—1W-AKE

In a multi-pass circuit construction, a session key is established via a Diffie–Hellman key exchange. However, the precise properties required of this protocol were not formalized until recently. Goldberg, Stebila and Ustaoglu [13] formalized the concept of 1W-AKE, presented an efficient instantiation, and described its utility towards onion routing. We review their work here and we refer the readers to [13] 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, as in Tor, the unauthenticated party may even want to preserve its anonymity. Their 1W-AKE protocol constitutes this precise primitive.

The 1W-AKE protocol consists of three procedures: *Initiate*, *Respond*, and *ComputeKey*. With procedure *Initiate*, Alice (or her onion proxy) generates and sends an authentication challenge to the server (an OR node). The OR node responds to the challenge by running the *Respond* procedure, and returning the authentication response. The onion proxy (OP) then runs the *ComputeKey* procedure over the received response to authenticate the OR node and compute the session key. A 1W-AKE protocol also satisfies 1W-AKE security and one way anonymity properties, which we leave to Section 4.3.

In terms of instantiation, Goldberg et al. show that an AKE protocol suggested for Tor—the fourth protocol in [26]—can be attacked, leading to an adversary determining all of the user's session keys.

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

They then fix the protocol (see Figure 14) and proved that the fixed protocol (ntor) satisfies the formal properties of 1W-AKE. In our OR analysis, we use their formal definition and their fixed protocol.

2.3 The OR Protocol

We describe an OR protocol Π_{OR} that follows the Tor specification [7]. We do not present the cryptographic algorithms, e.g., wrapping and unwrapping onions, in this section but only present the skeleton of the protocol. A thorough characterization of these cryptographic algorithms follows in Section 4.

We describe our protocols using pseudocode and assume that a node maintains a state for every execution and responds (changes the state and/or sends a message) upon receiving a message as per its current state.

There are two types of messages that the protocol generates and processes: The first type contains *input actions*, which carry inputs to the protocol from the user (Alice), and *output actions*, which carry output of the protocol to Alice. The second message type is a point-to-point *network message* (a cell in the OR literature), which is to be delivered by one protocol node to another. To enter a wait state, a thread may execute a command of the form **wait for** a network message.

With this methodology, we are able to effortlessly extract an OR protocol (Π_{OR}) from the Tor specification by categorizing actions based on the OR cell types (see Figure 1). For ease of exposition, we only consider Tor cells that are cryptographically important and relevant from the security definitional perspective. In particular, we consider create, created and destroy cells among control cells, and data, extend and extended cells among relay cells. We also include two input messages createcircuit and send, where Alice uses createcircuit to create OR circuits and uses send to send messages m over already-created circuits. We do not consider streams and the SOCKS interface in Tor as they are extraneous to the basic OR methodology. We unify instructions for an OP node and an OR node for the simplicity of discussion. Moreover, for the sake of brevity, we restrict ourselves to messages $m \in M(\kappa)$ that fit exactly in one cell. It is straight-forward to extend our result to a protocol that accepts larger messages. The only difference is that the onion proxy and the exit node divide message into smaller pieces and recombine them in an appropriate way.

Function calls *Initiate*, *Respond* and *ComputeKey* correspond to 1W-AKE function calls described in Section 2.2. Function calls *WrOn* and *UnwrOn* correspond to the principal onion algorithms. *WrOn* creates a layered encryption of a payload (plaintext or onion) for given an ordered list of ℓ session keys for $\ell \geq 1$. *UnwrOn* removes ℓ layers of encryptions from an onion to output a plaintext or an onion given an input onion and a ordered list of ℓ session keys for $\ell \geq 1$. Moreover, onion algorithms also ensure end-to-end integrity. The cryptographic requirements for these onion algorithms are presented in Section 4.2.

Tor uses a centralized approach to determine valid OR nodes and distribute their public keys. Every OR node has to be registered in so-called directory servers, where each registration is checked by an administrator. These directory servers then distribute the list of valid OR nodes and the respective public keys. We abstract these directory servers as an ideal service, formalized by a bulletin board functionality $\mathcal{F}_{\text{REG}}$ (see Canetti [4]). Tor does not guarantee any anonymity once these directory servers are compromised. Therefore, we concentrate on the case in which these directory servers cannot be compromised.[2] As in Tor, we assume that the list of valid OR nodes is given to the directory servers from outside, in our case from the environment. However, for the sake of simplicity we assume that the OR list is only synchronized initially. In detail, we slightly extend the functionality as follows. $\mathcal{F}_{\text{REG}}$ initially receives a list of OR nodes from the environment, waits for each of these parties for a public key, and distributes the list of OR nodes and their public keys as $(\mathsf{registered}, \langle P_j, pk_j \rangle_{j=1}^{n})$. Each OR node, on the other hand, initially computes its long-term keys (sk, pk) and registers the public part at $\mathcal{F}_{\text{REG}}$. Then, the node waits to receive the message $(\mathsf{registered}, \langle P_j, pk_j \rangle_{j=1}^{n})$ from $\mathcal{F}_{\text{REG}}$ before declaring that it is ready for use.[3]

OPs develop circuits incrementally, one hop at a time, using the *ExtendCircuit* function defined in Figure 2. To create a new circuit, an OP sends a create cell to the first node, after calling the *Initiate* function of 1W-AKE; the first node responds with a created cell after running the *Respond* function. The OP then runs the *ComputeKey* function. To extend a circuit past the first node, the OP sends an extend relay cell after calling the *Initiate* function, which instructs the last node in the circuit to send a create cell to extend the circuit.

[2] Formally, this ideal functionality $\mathcal{F}_{\text{REG}}$ does not accept compromise-requests from the attacker.

[3] The functionality $\mathcal{F}_{\text{REG}}$ additionally answers upon a request retrieve with the full list of participants $\langle P_j, pk_j \rangle_{j=1}^{n}$.

upon an input (setup):
  Generate an asymmetric key pair $(sk, pk) \leftarrow G$.
  send a cell (register, P, pk) to the $\mathcal{F}_{\text{REG}}$ functionality
  wait for a cell (registered, $\langle P_j, pk_j\rangle_{j=1}^{n}$) from $\mathcal{F}_{\text{REG}}$
  output (ready, $\mathcal{N} = \langle P_j\rangle_{j=1}^{n}$)
upon an input (createcircuit, $\mathcal{P} = \langle P, \langle P_j\rangle_{j=1}^{\ell}\rangle$):
  store $\mathcal{P}$ and $\mathcal{C} \leftarrow \langle P\rangle$; call $ExtendCircuit(\mathcal{P}, \mathcal{C})$
upon an input (send, $\mathcal{C} = \langle P \overset{cid_1}{\Longleftrightarrow} P_1 \Longleftrightarrow \cdots P_\ell\rangle, m$):
  if $Used(cid_1) < ttl_C$ **then**
    look up the keys $(\langle k_j\rangle_{j=1}^{\ell}$ for cid_1
    $O \leftarrow WrOn(m, (k_j)_{j=1}^{\ell})$; $Used(cid_1){+}{+}$
    send a cell (cid_1, relay, O) to P_1 over $\mathcal{F}_{\text{SCS}}$
  else
    call $DestroyCircuit(\mathcal{C}, cid_1)$; output (destroyed, $\mathcal{C}, m$)
upon receiving a cell (cid, create, X) from P_i over $\mathcal{F}_{\text{SCS}}$:
  $\langle Y, k_{\text{new}}\rangle \leftarrow Respond(pk_P, sk_P, X)$
  store $\mathcal{C} \leftarrow \langle P_i \overset{cid, k_{\text{new}}}{\Longleftrightarrow} P\rangle$
  send a cell (cid, created, Y, t) to P_i over $\mathcal{F}_{\text{SCS}}$
upon receiving a cell (cid, created, Y, t) from P_i over $\mathcal{F}_{\text{SCS}}$:
  if $prev(cid) = (P', cid', k')$ **then**
    $O \leftarrow WrOn(\langle \text{extended}, Y, t\rangle, k')$
    send a cell (cid', relay, O) to P' over $\mathcal{F}_{\text{SCS}}$
  else if $prev(cid) = \perp$ **then**
    $k_{\text{new}} \leftarrow ComputeKey(pk_i, Y, t)$
    update $\mathcal{C}$ with k_{new}; call $ExtendCircuit(\mathcal{P}, \mathcal{C})$
upon receiving a cell (cid, relay, O) from P_i over $\mathcal{F}_{\text{SCS}}$:
  if $prev(cid) = \perp$ **then**
    if $getkey(cid) = (k_j)_{j=1}^{\ell'}$ **then**
      (type, m) **or** $O \leftarrow UnwrOn(O, (k_j)_{j=1}^{\ell'})$
      (P', cid') **or** $\perp \leftarrow next(cid)$
  else if $prev(cid) = (P', cid', k')$ **then**
    $O \leftarrow WrOn(O, k')$ /* a backward onion */
  switch (type)
  case extend:
    get $\langle P_{next}, X\rangle$ from m; $cid_{next} \overset{\$}{\leftarrow} \{0,1\}^\kappa$
    update $\mathcal{C} \leftarrow \langle P_i \overset{cid, k}{\Longleftrightarrow} P \overset{cid_{next}}{\Longleftrightarrow} P_{next}\rangle$
    send a cell (cid_{next}, create, X) to P_{next} over $\mathcal{F}_{\text{SCS}}$
  case extended:
    get $\langle Y, t\rangle$ from m; get P_{ex} from $(\mathcal{C}, \mathcal{P})$
    $k_{\text{ex}} \leftarrow ComputeKey(pk_{\text{ex}}, Y, t)$
    update $\mathcal{C}$ with (k_{ex}); call $ExtendCircuit(\mathcal{P}, \mathcal{C})$
  case data:
    if (P = OP) **then** output (received, $\mathcal{C}, m$)
    else if $m = (S, sid, m')$ send (P, S, sid, m') to $\mathcal{F}_{\text{NET}^q}$
  case *corrupted*: /*corrupted onion*/
    call $DestroyCircuit(\mathcal{C}, cid)$
  case default: /*encrypted forward/backward onion*/
    send a cell (cid', relay, O) to P' over $\mathcal{F}_{\text{SCS}}$
upon receiving a msg (sid, m) from $\mathcal{F}_{\text{NET}^q}$:
  get $\mathcal{C} \leftarrow \langle P' \overset{cid, k}{\Longleftrightarrow} P\rangle$ for sid; $O \leftarrow WrOn(m, k)$
  send a cell (cid, relay, O) to P' over $\mathcal{F}_{\text{SCS}}$
upon receiving a cell (cid, destroy) from P_i over $\mathcal{F}_{\text{SCS}}$:
  call $DestroyCircuit(\mathcal{C}, cid)$

Figure 1: Π_{OR}: The OR Protocol for Party P

Circuits are identified by circuit IDs ($cid \in \{0,1\}^\kappa$) that associate two consecutive circuit nodes. We denote circuit at a node P_i using the terminology $\mathcal{C} = P_{i-1} \overset{cid_i, k_i}{\Longleftrightarrow} P_i \overset{cid_{i+1}}{\Longleftrightarrow} P_{i+1}$, which says that

$\mathit{ExtendCircuit}(\mathcal{P} = \langle P_j \rangle_{j=1}^{\ell}, \mathcal{C} = \langle P \overset{cid_1,k_1}{\Longleftrightarrow} P_1 \overset{k_2}{\Longleftrightarrow} \cdots P_{\ell'} \rangle)$:
  determine the next node $P_{\ell'+1}$ from $\mathcal{P}$ and $\mathcal{C}$
  if $P_{\ell'+1} = \bot$ **then**
    output $(\mathsf{created}, \langle P \overset{cid_1}{\Longleftrightarrow} P_1 \Longleftrightarrow \cdots P_{\ell'} \rangle)$
  else
    $X \leftarrow \mathit{Initiate}(pk_{P_{\ell'+1}}, P_{\ell'+1})$
    if $P_{\ell'+1} = P_1$ **then**
      $cid_1 \xleftarrow{\$} \{0,1\}^{\kappa}$
      send a cell $(cid_1, \mathsf{create}, X)$ to P_1 over $\mathcal{F}_{\mathrm{SCS}}$
    else
      $O \leftarrow \mathit{WrOn}(\{\mathsf{extend}, P_{\ell'+1}, X\}, (k_j)_{j=1}^{\ell'})$
      send a cell $(cid_1, \mathsf{relay}, O)$ to P_1 over $\mathcal{F}_{\mathrm{SCS}}$

$\mathit{DestroyCircuit}(\mathcal{C}, cid)$:
  if $\mathit{next}(cid) = (P_{next}, cid_{next})$ **then**
    send a cell $(cid_{next}, \mathsf{destroy})$ to P_{next} over $\mathcal{F}_{\mathrm{SCS}}$
  else if $\mathit{prev}(cid) = (P_{prev}, cid_{prev})$ **then**
    send a cell $(cid_{prev}, \mathsf{destroy})$ to P_{prev} over $\mathcal{F}_{\mathrm{SCS}}$
  discard $\mathcal{C}$ and all streams

Figure 2: Subroutines of Π_{OR} for Party P

P_{i-1} and P_{i+1} are respectively the predecessor and successor of P_i in a circuit $\mathcal{C}$. k_i is a session key between P_i and the OP, while the absence of k_{i+1} indicates that a session key between P_{i+1} and the OP is not known to P_i; analogously the absence of a circuit id *cid* in that notation means that only the first circuit id is known, as for OP, for example. Functions *prev* and *next* on *cid* correspondingly return information about the predecessor or successor of the current node with respect to *cid*; e.g., $\mathit{next}(cid_i)$ returns (P_{i+1}, cid_{i+1}) and $\mathit{next}(cid_{i+1})$ returns $\bot$. The OP passes on to Alice $\langle P \overset{cid_1}{\Longleftrightarrow} P_1 \Longleftrightarrow \cdots P_{\ell} \rangle$.

Within a circuit, the OP and the exit node use relay cells created using *WrOn* to tunnel end-to-end commands and connections. The exit nodes use some additional mechanisms (streams used in Tor) to synchronize communication between the network and a circuit $\mathcal{C}$. We represent that using *sid*. With this auxiliary synchronization, end-to-end communication between OP and the exit node happens with a *WrOn* call with multiple session keys and a series of *UnwrOn* calls with individual session keys in the forward direction, and a series of *WrOn* calls with individual session keys, and finally a *UnwrOn* call with multiple session keys in the backward direction. Communication in the forward direction is initiated by a send message by Alice to the OP, while communication in the backward direction is initiated by a network message to the exit node. Cells are exchanged between OR nodes over a secure and authenticated channels, e.g., a TLS connection. As proposed by Canetti, we abstract such a channel in the UC framework by a functionality $\mathcal{F}_{\mathrm{SCS}}$ [4].[4]

To tear down a circuit completely, an OR or OP sends a destroy cell to the adjacent nodes on that circuit with appropriate *cid* using the *DestroyCircuit* function defined in Figure 2. Upon receiving an outgoing destroy cell, a node frees resources associated with the corresponding circuit. If it is not the end of the circuit, it sends a destroy cell to the next node in the circuit. Once a destroy cell has been processed, the node ignores all cells for the corresponding circuit. Note that if an integrity check fails during *UnwrOn*, the destroy cells are sent in the forward and backward directions in a similar way.

In the Tor the OP has a time limit (of ten minutes) for each established circuit; thereafter, the OP constructs a new circuit. However, the UC framework does not provide a notion of time. We model such a time limit in the UC framework by only allowing a circuit to transport at most a constant number (say ttl_C) of messages measured using the *used* function call. Afterwards, the OP discards the circuit and establishes a fresh circuit.

2.4 The UC Framework: An Overview

The UC framework is designed to enable a modular analysis of security protocols. In this framework, the security of a protocol is defined by comparing it with a setting in which all parties have a direct and private connection to a trusted machine that computes the desired functionality. As an example consider an authenticated channel between Alice and Bob with a passive attacker. In the real world Alice would

[4]As leakage function l for $\mathcal{F}_{\mathrm{SCS}}$, we choose $l(m) := |m|$.

upon receiving a msg $(\mathsf{compromise}, \mathsf{N}_\mathcal{A})$ from $\mathcal{A}$:
  set $compromised(P) \leftarrow true$ **for** every $P \in \mathsf{N}_\mathcal{A}$
  set $b \leftarrow \frac{|\mathsf{N}_\mathcal{A}|}{|\mathsf{N}_{\mathrm{OR}}|}$
upon an input $(\mathsf{send}, S, [m])$ from the environment for party U:
  with probability b^2,
    choose $P_\ell \xleftarrow{\$} \mathsf{N}_\mathcal{A}$
    send $(\mathsf{sent}, U, S, [m])$ to $\mathcal{A}$
  with probability $(1-b)b$,
    choose $P_\ell \xleftarrow{\$} \mathsf{N}_\mathcal{A}$
    send $(\mathsf{sent}, -, S, [m])$ to $\mathcal{A}$
  with probability $b(1-b)$,
    choose $P_\ell \xleftarrow{\$} \mathsf{N}_{\mathrm{OR}} \setminus \mathsf{N}_\mathcal{A}$
    send $(\mathsf{sent}, U, -)$ to $\mathcal{A}$
  with probability $(1-b)^2$,
    choose $P_\ell \xleftarrow{\$} \mathsf{N}_{\mathrm{OR}} \setminus \mathsf{N}_\mathcal{A}$
    send $(\mathsf{sent}, -, -)$ to $\mathcal{A}$
  output message $(P_\ell, S, [m])$

Figure 3: Black-box OR Functionality $\mathcal{B}_{\mathrm{OR}}$ [11]

call a protocol that signs the message m to be communicated and sends the signed message over the network such that Bob would verify the signature. In the setting with a trusted machine T, however, Alice sends the message m directly to T[5]; T notifies the attacker about m, and T directly sends m to Bob. This trusted machine is called the *ideal functionality.*

Security in the UC framework is defined as follows: A protocol π *UC-realizes* an ideal functionality $\mathcal{F}$ if for all probabilistic poly-time (PPT) attackers $\mathcal{A}$ there is a PPT simulator S such that no PPT machine can distinguish an interaction with π and $\mathcal{A}$ from an interaction with $\mathcal{F}$ and S. The distinguisher is connected to the protocol and the attacker (or the simulator).

2.5 An OR Black Box Model

Anonymity in a low-latency OR network does not only depend upon the security of the onions but also upon the magnitudes and distributions of users and their destination servers. In the OR literature, considerable efforts have been put towards measuring the anonymity of onion routing [9–11, 23, 30].

Feigenbaum, Johnson, and Syverson used for an analysis of the anonymity properties of onion routing an ideal functionality $\mathcal{B}_{\mathrm{OR}}$ [11]. This functionality emulates an I/O-automata model for onion routing from [9, 10]. Figure 3 presents this functionality $\mathcal{B}_{\mathrm{OR}}$.

Let N_{OR} be the set of onion routers, and let $\mathsf{N}_\mathcal{A}$ of those be eavesdropped, where $b = |\mathsf{N}_\mathcal{A}|/|\mathsf{N}_{\mathrm{OR}}|$ defines the fraction of compromised nodes. It takes as input from each user U the identity of a destination S. For every such connection between a user and a destination, the functionality may reveal to the adversary the identity of the user $(\mathsf{sent}, U, -)$ (i.e., the first OR router is compromised), the identity of the destination $(\mathsf{sent}, -, S, [m])$ (i.e., the exit node is compromised), both $(\mathsf{sent}, U, S, [m])$ (i.e., the first OR router and the exit node are compromised) or only a notification that something has been sent $(\mathsf{sent}, -, -)$ (i.e., neither the first OR router not the exit node is compromised).

We stress that this functionality only abstracts an OR network against local attackers.As the distribution of the four cases only depends on the first and the last router being compromised but not on the probability that the attacker controls sensitive links between honest parties, $\mathcal{B}_{\mathrm{OR}}$ only models OR against local adversaries. As an example consider, the case in which the attacker only wiretaps the connection between the exit node and the server. In this case, the attacker is able to determine which message has been sent to whom, i.e., the abstraction needs to leak $(\mathsf{sent}, -, S, [m])$; however, the probability of this event is c, where c is the fraction of observed links between honest onion routers and users and servers. Therefore, $\mathcal{B}_{\mathrm{OR}}$ cannot be used as an abstraction for onion routing against partially global attackers. In Section 7.1.1, we present an extension of $\mathcal{B}_{\mathrm{OR}}$ that models onion routing against partially global attackers and prove that it constitutes a sound abstraction.

We actually present $\mathcal{B}_{\mathrm{OR}}$ in two variants. In the first variant $\mathcal{B}_{\mathrm{OR}}$ does not send an actual message but only a notification. This variant has been analyzed by Feigenbaum, Johnson, and Syverson. We

[5] Recall that T and Alice are directly connected, as well as T and Bob.

additionally consider the variant in which $\mathcal{B}_{\text{OR}}$ sends a proper message m. We denote these two variants by marking the message m as optional, i.e., as $[m]$.

In order to justify these OR anonymity analyses that consider an OR network as a black box, it important to ascertain that these black boxes indeed model onion routing. In particular, it is important under which adversary and network assumptions these black boxes model real-world OR networks. In this work, we show that the black box $\mathcal{B}_{\text{OR}}$ can be UC-realized by a simplified version of the Tor network.

3 Security Definition of OR

In this section, we first describe our system and adversary model for all protocol that we analyze (Section 3.1). Thereafter, we present a composable security definition of OR by introducing an ideal functionality (abstraction) $\mathcal{F}_{\text{OR}}$ in the UC framework (Section 3.2).

Tor was designed to guarantee anonymity even against partially global attacker, i.e., attacker that do not only control compromised OR nodes but also a portion of the network. Previous work, however, only analyzed local, static attackers [9–11], such as the abstraction $\mathcal{B}_{\text{OR}}$ presented in Figure 3). In contrast, we analyze onion routing against partially global attackers. As our resulting abstraction $\mathcal{F}_{\text{OR}}$ has to faithfully reflect that an active attacker can hold back all onions that it observes, $\mathcal{F}_{\text{OR}}$ is naturally more complex than $\mathcal{B}_{\text{OR}}$.

3.1 System and Adversary Model

We consider a fully connected network of n parties $\mathsf{N} = \{P_1, \dots, P_n\}$. For simplicity of presentation, we consider all parties to be OR nodes that also can function as OPs to create circuits and send messages. It is also possible to use our formulation to model separate user OPs that only send and receive messages but do not relay onions.

Tor has not been designed to resist against global attackers. Such an attacker is too strong for many practical purposes as it can simply break the anonymity of an OR protocol by holding back all but one onion and tracing that one onion though the network. However, in contrast to previous work, we do not only consider local attackers, which do not control more than the compromised OR routers, but also partially global attackers that control a certain portion of the network. Analogous to the network functionality $\mathcal{F}_{\text{SYN}}$ proposed by Canetti [4], we model the network as an ideal functionality $\mathcal{F}_{\text{NET}^q}$, which bounds the number of attacker-controlled links to $q \in [0, \binom{n}{2})]$. For attacker-controlled links the messages are forwarded to the attacker; otherwise, they are directly delivered. In Section 7 we show that previous black-box analyses of onion routing against local attackers applies to our setting as well by choosing $q := 0$. Let S represent all possible destination servers $\{S_1, \dots, S_\Delta\}$ which reside in the network abstracted by a network functionality $\mathcal{F}_{\text{NET}^q}$.

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

Adaptive Corruptions. Forward secrecy [6] is an important property for onion routing. In order to analyze this property, we allow adaptive corruptions of nodes by the attacker $\mathcal{A}$. Such an adaptive corruption is formalized by a message compromise, which is sent to the respective party. Upon such a compromise message the internal state of that party is deleted and a long-term secret key sk for the node is revealed to the attacker. $\mathcal{A}$ can then impersonate the node in the future; however, $\mathcal{A}$ cannot obtain the information about its ongoing sessions. We note that this restriction arises due to the currently available security proof techniques and the well-known selective opening problem with symmetric encryptions [18], and the restriction is not specific to our constructions [2, 14]. We could also restrict ourselves to a static adversary as in previous work [3]; however, that would make an analysis of forward secrecy impossible.

3.2 Ideal Functionality

The presentation of the ideal functionality $\mathcal{F}_{\text{OR}}$ is along the lines of the description OR protocol Π_{OR} from Section 2.3. We continue to use the message-based state transitions from Π_{OR}, and consider sub-machines for all n nodes in the ideal functionality. To communicate with each other through messages and data structures, these sub-machines share a memory space in the functionality. The sub-machine pseudocode for the ideal functionality appears in Figure 4 and three subroutines are defined in Figure 5. As the similarity between pseudocodes for the OR protocol and the ideal functionality is obvious, rather than explaining the OR message flows again, we concentrate on the differences.

upon an input (setup):
  $SendMessage(\mathsf{dir}, \mathsf{register}, P)$
  wait for a msg $(\mathsf{dir}, \mathsf{registered}, \langle P_j \langle_{j=1}^{n} \rangle)$ via a handle
  output $(\mathsf{ready}, \mathcal{N} = (P_j)_{j=1}^{n})$
upon an input $(\mathsf{createcircuit}, \mathcal{P} = \langle P, P_1, \ldots, P_\ell \rangle)$:
  store $\mathcal{P}$ and $\mathcal{C} \leftarrow \langle P \rangle$; $ExtendCircuit(\mathcal{P}, \mathcal{C})$
upon an input $(\mathsf{send}, \mathcal{C} = \langle P \overset{cid_1}{\Longleftrightarrow} P_1 \Longleftrightarrow \cdots P_\ell \rangle, m)$:
  if $Used(cid_1) < ttl_C$ **then**
    $Used(cid_1)$++; $SendMessage(P_1, cid_1, \mathsf{relay}, \langle \mathsf{data}, m \rangle)$
  else
    $DestroyCircuit(\mathcal{C}, cid_1)$; output $(\mathsf{destroyed}, \mathcal{C}, m)$
upon receiving a handle $\langle P, P_{next}, h \rangle$ from $\mathcal{F}_{\text{NET}^q}$:
  send $(msg) \leftarrow lookup(h)$ to a receiving submachine P_{next}
upon receiving a msg $(P_i, cid, \mathsf{create})$ through a handle:
  store $\mathcal{C} \leftarrow \langle P_i \overset{cid}{\Longleftrightarrow} P \rangle$
  $SendMessage(P_i, cid, \mathsf{created})$
upon receiving a msg $(P_i, cid, \mathsf{created})$ through a handle:
  if $prev(cid) = (P', cid')$ **then**
    $SendMessage(P', cid', \mathsf{relay}, \mathsf{extended})$
  else if $prev(cid) = \perp$ **then**
    $ExtendCircuit(\mathcal{P}, \mathcal{C})$
upon receiving a msg $(P_i, cid, \mathsf{relay}, O)$ through a handle:
  if $prev(cid) = \perp$ **then**
    if $next(cid) = \perp$ **then**
      get (type, m) from O
    else $\{P', cid'\} \leftarrow next(cid)$
  else
    $(P', cid') \leftarrow prev(cid)$
  switch (type)
  case extend:
    get P_{next} from m; $cid_{next} \overset{\$}{\leftarrow} \{0,1\}^\kappa$
    update $\mathcal{C} \leftarrow \langle P_i \overset{cid}{\Longleftrightarrow} P \overset{cid_{next}}{\Longleftrightarrow} P_{next} \rangle$
    $SendMessage(P_{next}, cid_{next}, \mathsf{create})$
  case extended:
    update $\mathcal{C}$ with P_{ex}; $ExtendCircuit(\mathcal{P}, \mathcal{C})$
  case hidden services:
    output m /*Further processing at a higher level*/
  case data:
    if (P = OP) **then** output $(\mathsf{received}, C, m)$
    else if $m = (S, sid, m')$ send (P, S, sid, m') to $\mathcal{F}_{\text{NET}^q}$
  case *corrupted*: /*corrupted onion*/
    $DestroyCircuit(\mathcal{C}, cid)$
  case default: /*encrypted forward/backward onion*/
    $SendMessage(P', cid', \mathsf{relay}, O)$
upon receiving a msg (sid, m) from $\mathcal{F}_{\text{NET}^q}$:
  obtain $\mathcal{C} = \langle P' \overset{cid}{\Longleftrightarrow} P \rangle$ for sid
  $SendMessage(P', cid, \mathsf{relay}, \langle \mathsf{data}, m \rangle)$
upon receiving a msg $(P_i, cid, \mathsf{destroy})$ through a handle:
  $DestroyCircuit(\mathcal{C}, cid)$
upon receiving a msg$(P_i, P, h, [\mathsf{corrupt}, T(\cdot)])$from $\mathcal{A}$:
  $(msg) \leftarrow lookup(h)$
  if corrupt = *true* **then**
    $msg \leftarrow T(msg)$; set $corrupted(msg) \leftarrow true$
  process msg as the receiving submachine is P
upon receiving a msg $(\mathsf{compromise}, P)$ from $\mathcal{A}$:
  set $compromised(P) \leftarrow true$
  delete all local information at P

Figure 4: The ideal functionality $\mathcal{F}_{\text{OR}}$ for Party P

$ExtendCircuit(\mathcal{P} = (P_j)_{j=1}^{\ell}, \mathcal{C} = \langle P \overset{cid_1}{\Longleftrightarrow} P_1 \Longleftrightarrow \cdots P_{\ell'}\rangle)$:
  determine the next node $P_{\ell'+1}$ from $\mathcal{P}$ and $\mathcal{C}$
  if $P_{\ell'+1} = \perp$ **then**
    output $(\mathsf{created}, \mathcal{C})$
  else
    if $P_{\ell'+1} = P_1$ **then**
      $cid_1 \xleftarrow{\$} \{0,1\}^{\kappa}$; $SendMessage(P_1, cid_1, \mathsf{create})$
    else
      $SendMessage(P_1, cid_1, \mathsf{relay}, \{\mathsf{extend}, P_{\ell'+1}\})$

$DestroyCircuit(\mathcal{C}, cid)$:
  if $next(cid) = (P_{next}, cid_{next})$ **then**
    $SendMessage(P_{next}, cid_{next}, \mathsf{destroy})$
  else if $prev(cid) = (P_{prev}, cid_{prev})$ **then**
    $SendMessage(P_{prev}, cid_{prev}, \mathsf{destroy})$
  discard $\mathcal{C}$ and all streams

$SendMessage(P_{next}, cid_{next}, \mathsf{cmd}, [\mathsf{relay\text{-}type}], [\mathsf{data}])$:
  create a msg for P_{next} from the input
  draw a fresh handle h and set $lookup(h) \leftarrow msg$
  if $compromised(P_{next}) = true$ **then**
    P_{last} is the last node in the complete continuous *compromised* path starting P_{next}
    if $(P_{last} = OP)$ or P_{last} is the exit node **then**
      send the entire msg to $\mathcal{A}$
    else
      send $\langle P, P_{next}, \ldots, P_{last}, cid_{next}, \mathsf{cmd}, h\rangle$ to $\mathcal{A}$
  else
    send $\langle P, P_{next}, h\rangle$ to the network

Figure 5: Subroutines of $\mathcal{F}_{\text{OR}}$ for Party P

The only major difference between Π_{OR} and $\mathcal{F}_{\text{OR}}$ is that cryptographic primitives such as message wrapping, unwrapping, and key exchange are absent in the ideal world; we do not have any keys in $\mathcal{F}_{\text{OR}}$, and the OR messages *WrOn* and *UnwrOn* as well as the 1W-AKE messages *Initiate*, *Respond*, and *ComputeKey* are absent.

The ideal functionality also abstracts the directory server and expects on the input/output interface of $\mathcal{F}_{\text{REG}}$ (from the setting with Π_{OR}) an initial message with the list $\langle P_i\rangle_{i=1}^{n}$ of valid nodes. This initial message corresponds to the list of onion routers that have been approved by an administrator. We call the part of $\mathcal{F}_{\text{OR}}$ that abstracts the directory servers dir. For the sake of brevity, we do not present the pseudocode of dir. Upon an initial message with a list $\langle P_i\rangle_{i=1}^{n}$ of valid nodes, dir waits for all nodes P_i ($i \in \{1, \ldots, n\}$) for a message $(\mathsf{register}, P_i)$. Once all nodes registered, dir sends a message $(\mathsf{registered}, \langle P_i\rangle_{i=1}^{n})$ with a list of valid and registered nodes to every party that registered, and to every party that sends a retrieve message to dir.

Messages from $\mathcal{A}$ and $\mathcal{F}_{\text{NET}^q}$. In Figure 4 and Figure 6, we present the pseudocode for the attacker messages and the network functionality, respectively. For our basic analysis, we model an adversary that can control all communication links and servers in $\mathcal{F}_{\text{NET}^q}$, but cannot view or modify messages between parties due to the presence of the secure and authenticated channel. Therefore, sub-machines in the functionality store their messages in the shared memory, and create and send handles $\langle P, P_{next}, h\rangle$ for these messages $\mathcal{F}_{\text{NET}^q}$. The message length does not need to be leaked as we assume a fixed message size (for all $M(\kappa)$). Here, P is the sender, P_{next} is the receiver and h is a handle or a pointer to the message in the shared memory of the ideal functionality. In our analysis, all $\mathcal{F}_{\text{NET}^q}$ messages flow to $\mathcal{A}$, which may choose to return these handles back to $\mathcal{F}_{\text{OR}}$ through $\mathcal{F}_{\text{NET}^q}$ at its own discretion. However, $\mathcal{F}_{\text{NET}^q}$ also maintains a mechanism through *observedLink* flags for the non-global adversary $\mathcal{A}$. The adversary may also corrupt or replay the corresponding messages; however, these active attacks are always detected by the receiver due to the presence of a secure and authenticated channel between any two communicating parties and we need not model these corruptions.

The adversary can compromise a party P or server S by sending a compromise message to respectively $\mathcal{F}_{\text{OR}}$ and $\mathcal{F}_{\text{NET}^q}$. For party P or server S, the respective functionality then sets *compromised* tag to *true*. Furthermore, all input or network messages that are supposed to be visible to the compromised entity

upon receiving a msg (obverse, P, P_{next}) from $\mathcal{A}$:
set $observedLink(P, P_{next}) \leftarrow true$
upon receiving a msg (compromise, S) from $\mathcal{A}$:
set $compromised(S) \leftarrow true$; send $\mathcal{A}$ all existing sid
upon receiving a msg $(P, P_{next}/S, m)$ from $\mathcal{F}_{\text{OR}}$:
if P_{next}/S is a $\mathcal{F}_{\text{OR}}$ node **then**
if $observedLink(P, P_{next}) = true$ **then**
forward the msg (P, P_{next}, m) to $\mathcal{A}$
else
reflect the msg (P, P_{next}, m) to $\mathcal{F}_{\text{OR}}$
else if P_{next}/S is a $\mathcal{F}_{\text{NET}^q}$ server **then**
if $compromised(S) = true$ **then**
forward the msg (P, S, m) to $\mathcal{A}$
else
output (P, S, m)
upon receiving a msg $(P/S, P_{next}, m)$ from $\mathcal{A}$:
forward the msg $(P/S, P_{next}, m)$ to $\mathcal{F}_{\text{OR}}$

Figure 6: The Network Functionality $\mathcal{F}_{\text{NET}^q}$

are forwarded to the adversary. In principle, the adversary runs that entity for the rest of the protocol and can send messages from that entity. In that case, it can also propagate corrupted messages which in Π_{OR} can only be detected during *UnwrOn* calls at OP or the exit node. We model these corruptions using $corrupted(msg) = \{true, false\}$ status flags, where $corrupted(msg)$ status of messages is maintained across nodes until they reach end nodes. Furthermore, for every corrupted message, the adversary also provides a modification function $T(\cdot)$ as the end nodes run by the adversary may continue execution even after observing a *corrupted* flag. In that case, $T(\cdot)$ captures the exact modificaiton made by the adversary.

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.[6] Similar to the previous work [3], a message is directly given to the adversary if all remaining nodes in a communication path are under adversary control.

4 Secure OR modules

We identify the core cryptographic primitives for a secure OR protocol. 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 consist of two parts: first, secure onion algorithm, and second, a one-way authenticated key-exchange primitive (1W-AKE), a notion recently introduced by Goldberg, Stebila, and Ustaoglu [13].

Onion algorithms typically use several layers of encryptions and possibly integrity mechanisms, such as message authentication codes. Previous attempts [3] for proving the security OR protocols use mechanisms to ensure hop-to-hop integrity, such as non-malleable encryption schemes. The widely-used Tor network, however, does not use hop-to-hop integrity but only end-to-end integrity. In the analysis of OR protocols with only end-to-end integrity guarantees, we also have to consider the cases in which the end node is compromised, thus no integrity check is performed at all. In order to cope with these cases, we identify a new notion of predictably malleable encryption schemes. Predictable malleability allows the attacker to change the ciphertexts but requires the resulting changes to the plaintext to be efficiently predictable given only the changes of the ciphertext.In Section 4.1 we rigorously define the notion of *predictably malleable* encryption schemes.

Inspired by Section 4.1, we introduce in Section 4.2 the notion of *secure onion algorithms*. In Section 4.3, we review the notion of one-way authenticated key-exchange (1W-AKE), which requires that the key-exchange protocol is *one-way authenticated*, i.e., the receiver cannot be impersonated, and *anonymous*, i.e., the sender cannot be identified.

[6] 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 (initialize)
$k \leftarrow \mathrm{G}(1^\eta)$
$s_d \leftarrow \varepsilon$
$s_e \leftarrow \varepsilon$

upon (encrypt, m)
if $b = 0$ **then**
$(c, s) \leftarrow \mathrm{E}(0^{|m|}, s_e, k)$
if $q(s_e) \neq \bot$ **then**
$(d, u) \leftarrow q(s_e)$
$c \leftarrow d$
else if $b = 1$ **then**
$(c, s) \leftarrow \mathrm{E}(m, s_e, k)$
$q(s_e) \leftarrow (c, m)$
$s_e \leftarrow s$
respond c

upon (decrypt, c)
$(d, u) \leftarrow q(s_d)$
$T \leftarrow \mathrm{D}(c, d)$
if $b = 0$ **then**
if $q(s_d) = \bot$ **then**
$(m, s) \leftarrow \mathrm{D}(c, s_d, k)$
$q(s_d) \leftarrow (c, m)$
else if $q(s_d) \neq \bot$ **then**
$m \leftarrow T(u)$
else if $b = 1$ **then**
$(m, s) \leftarrow \mathrm{D}(c, s_d, k)$
$s_d \leftarrow s$
respond (m, T)

Figure 7: The IND-PM Challenger PM-$\mathrm{Ch}_b^{\mathcal{E}}$

In the following definitions, we assume the PPT machines to actually be oracle machines. We write A^B to denote that A has oracle access to B.

4.1 Predictably Malleable Encryption

Simulation-based proofs often face their limits when dealing with malleable encryption. The underlying problem is that malleability induces an essentially arbitratrily large number of possibilities to modify ciphertexts, and the simulator has no possibility to predict the changes that are in fact going to happen.

We characterize the property of predicting the changes to the plaintext merely given the modifications on the ciphertext. Along the lines of the IND-CCA definition for stateful encryption schemes, we define the notion of *predictably malleable* (IND-PM) encryption schemes.[7] The attacker has access to an encryption and a decryption oracle, and either all encryption and decryption queries are honestly answered (the honest game) or all are faked (the faking game), i.e., $0^{|m|}$ is encrypted instead of a message m. In the faking game, the real messages are stored in some shared datastructure q, and upon a decryption query only look-ups in q are performed. The IND-PM challenger maintains a separate state, e.g., a counter, for encryption and decryption. These respective states are updated with each encryption decryption query.

In contrast to the IND-CCA challenger, the IND-PM challenger (see Figure 7) additionally stores the produced ciphertext together with the corresponding plaintext for each encryption query. Moreover, for each decryption call the challenger looks up the stored ciphertexts and messages. The honest decryption ignores the stored values and performs an honest decryption, but the faking decryption compares the stored ciphertext with the ciphertext from the query and tries to predict the modifications to the plaintext. Therefore, we require the existence of an efficiently computable algorithm D that outputs the description of an efficient transformation procedure T for the plaintext given the original ciphertext as well as the modified ciphertext.

Definition 1 (Predictable malleability)**.** *An encryption scheme $\mathcal{E} := (\mathrm{G}, \mathrm{E}, \mathrm{D})$ is IND-PM if there is a negligible function μ such that there is a deterministic polynomial-time algorithm* D *such that for all PPT attackers $\mathcal{A}$*

$$\Pr[b' \xleftarrow{\$} \{0,1\}, b \leftarrow \mathcal{A}(1^\kappa)^{\text{PM-Ch}_b^{\mathcal{E}}} : b = b'] \leq 1/2 + \mu(\kappa)$$

Moreover, we require that for all $m, c, s, k, k' \in \{0,1\}^*$

$$\Pr[(c', s') \leftarrow \mathrm{E}(m, k, s), \\ (m', s'') \leftarrow \mathrm{D}(c, k', s) : s' = s''] = 1$$

PM-$\mathrm{Ch}_0^{\mathcal{E}}$ *and* PM-$\mathrm{Ch}_1^{\mathcal{E}}$ *are defined in Figure 7.*

[7]The name predictable malleability is justified since it can be shown that every IND-CCA secure scheme is also IND-PM, and every IND-PM scheme in turn is IND-CPA secure. In the appendix, we show that detCTR is IND-PM.

We stress that the definition implies a super-polynomial length for state-cycles; otherwise there is in the faking game at least one repeated state s for which the two encrypt queries output the same ciphertext for any two plaintexts.

In Appendix 6.1, we show that deterministic counter-mode is IND-PM.

4.2 Secure Onion Algorithms

We identify the onion wrapping ($WrOn$) and unwrapping ($UnwrOn$) algorithms as central building blocks in onion routing. We identify four core properties of onion algorithms. The first property is *correctness*, i.e., if all parties behave honestly, the result is correct. The second property is the security of statefulness, coined *synchronicity*. It roughly states that whenever a wrapping and an unwrapping algorithms are applied to a message with unsynchronous states, the output is completely random. The third property is *end-to-end integrity*. The fourth property states that for all modifications to an onion the resulting changes in the ciphertext are predictable. We this property *predictable malleability*.

Onion Correctness. The first property of secure onion algorithms is *onion correctness*. It states that honest wrapping and unwrapping results in the same message. Moreover, the correctness states that whenever the unwrapping algorithm has a fake flag, it does not care about integrity, because for $m \in M(\kappa)$ the integrity measure is always added, as required by the end-to-end integrity. But for $m \notin M(\kappa)$ but of the right length, the wrapping is performed without an integrity measure. The fake flag then causes the unwrapping to ignore the missing integrity measure. Then, we also require that the state transition is independent from the message or the key.

Definition 2 (Onion correctness)**.** *Recall that $M(\kappa)$ is the message space for the security parameter κ. Let $\langle k_i \rangle_{i=1}^{\ell}$ be a sequence of randomly chosen bitstrings of length κ.*

Forward: $\Omega_f(m)$
$O_1 \leftarrow WrOn(m, \langle k_i \rangle_{i=1}^{\ell})$
for $i = 1$ *to* ℓ ***do***
$O_{i+1} \leftarrow UnwrOn(O_i, k_i)$
$x \leftarrow O_{\ell+1}$

Backward: $\Omega_b(m)$
$O_\ell \leftarrow WrOn(m, k_\ell)$
for $i = \ell - 1$ *to* 1 ***do***
$O_i \leftarrow WrOn(O_{i+1}, k_i)$
$x \leftarrow UnwrOn(O_1, \langle k_i \rangle_{i=1}^{\ell})$

Let Ω'_f be the defined as Ω_f except that $UnwrOn$ additionally uses the fake *flag. Analogously, Ω'_b is defined. We say that a pair of onion algorithms ($WrOn$, $UnwrOn$) is* correct *if the following three conditions hold:*

(i) $\Pr[x \leftarrow \Omega_d(m) : x = m] = 1$ *for* $d \in \{f, b\}$ *and* $m \in M(\kappa)$.

(ii) $\Pr[x \leftarrow \Omega_d(m) : x = m] = 1$ *for* $d \in \{f, b\}$ *and all* $m \in M'(\kappa) := \{m' | \exists m'' \in M(\kappa). |m'| = |m''|\}$.

(iii) For all $m \in M'(\kappa)$, $k, k' \in \{0,1\}^\kappa$ and $c, s \in \{0,1\}^*$ such that c is a valid onion and s is a valid state

$$\Pr[(c', s') \leftarrow WrOn(m, k, s),\\ (m', s'') \leftarrow UnwrOn(c, k', s) : s' = s''] = 1$$

(iv) $WrOn$ and $UnwrOn$ are polynomial-time computable and randomized algorithms.

Synchronicity. The second property is synchronicity. In order to achieve replay resistance, we have to require that once the wrapping and unwrapping do not have synchronized states anymore, the output of the wrapping and unwrapping algorithms is indistinguishable from randomness.

Definition 3 (Synchronicity)**.** *For a machine $\mathcal{A}$, let $\Omega_{l,\mathcal{A}}$ and $\Omega_{r,\mathcal{A}}$ be defined as follows:*

Left: $\Omega_{l,\mathcal{A}}(\kappa)$
$(m_1, m_2, st) \leftarrow \mathcal{A}(1^\kappa)$
$k, s, s' \xleftarrow{\$} \{0,1\}^\kappa$
$O \leftarrow WrOn(m_1, k, s)$
$O' \leftarrow UnwrOn(O, k, s')$
$b \leftarrow \mathcal{A}(O', st)$

Right: $\Omega_{r,\mathcal{A}}(\kappa)$
$(m_1, m_2, st) \leftarrow \mathcal{A}(1^\kappa)$
$k, s, s' \xleftarrow{\$} \{0,1\}^\kappa$
$O \leftarrow WrOn(m_2, k, s)$
$O' \leftarrow UnwrOn(O, k, s')$
$b \leftarrow \mathcal{A}(O', st)$

For all PPT machines $\mathcal{A}$ the following is negligible in κ:

$$|\Pr[b \leftarrow \Omega_{l,\mathcal{A}}(\kappa) : b = 1] - \Pr[b \leftarrow \Omega_{r,\mathcal{A}}(\kappa) : b = 1]|$$

(setup, ℓ')
if *initiated* = *false* **then**
  for $i = 1$ to ℓ' **do**
    $k_i \xleftarrow{\$} \{0,1\}^\kappa$; $cid_i \xleftarrow{\$} \{0,1\}^\kappa$
  initiated ← *true*; store ℓ'
  send *cid*

(compromise, i)
initiated ← *false*; erase the circuit
compromised(i) ← *true*; run setup;
for j with *compromised*(j) = *true* **do**
  send (cid_j, k_j) for all

(send, m)
$O \leftarrow WrOn(m, \langle k_i \rangle_{i=1}^{\ell'})$
send O

(unwrap, O, *cid*)
look up the key k for *cid*
$O' \leftarrow UnwrOn(O, k)$
send O'

(respond, m)
$O \leftarrow WrOn(m, k_{\ell'})$
send O

(wrap, O, *cid*)
look up the key k for *cid*
$O' \leftarrow WrOn(O, k)$
send O'

(destruct, O)
$m \leftarrow UnwrOn(O, \langle k_i \rangle_{i=1}^{\ell'})$
send m

Figure 8: The Honest Onion Secrecy Challenger $\mathsf{OS\text{-}Ch}^0$: $\mathsf{OS\text{-}Ch}^0$ only answers for honest parties

End-to-end integrity. The third property that we require is *end-to-end integrity*; i.e., the attacker is not able to produce an onion that successfully unwraps unless it compromises the exit node.For the following definition, we modify $\mathsf{OS\text{-}Ch}^0$ such that, along with the output of the attacker, also the state of the challenger is output. In turn, the resulting challenger $\mathsf{OS\text{-}Ch}^{0'}$ can optionally get a state s as input. In particular, $(a, s) \leftarrow A^B$ denotes in the following definition denotes the pair of the outputs of A and B.

For the following definition we use the modified challenger $\mathsf{OS\text{-}Ch}^{0'}$, which results from modifying $\mathsf{OS\text{-}Ch}^0$ such that along with the output of the attacker also the state of the challenger is output. The resulting challenger $\mathsf{OS\text{-}Ch}^{0'}$ can, moreover, optionally get a state s as input.

Definition 4 (End-to-end integrity). *Let $S(O, cid)$ be the machine that sends a* (destruct, O) *query to the challenger and outputs the response. Let $Q'(s)$ be the set of answers to* construct *queries from the challenger to the attacker. Let the last onion $O_{\ell'}$ of an onion O_1 be defined as follows:*

Last(O_1)*:*
  for $i = 1$ *to* $\ell' - 1$ ***do***
    $O_{i+1} \leftarrow UnwrOn(O_i)$

Let $Q(s) := \{\mathsf{Last}(O_1) \mid O_1 \in Q'(s)\}$ be the set of last onions answers to the challenger. We say a set of onion algorithms has end-to-end integrity *if for all PPT attackers $\mathcal{A}$ the following is negligible in κ*

$$\Pr[(O, s) \leftarrow \mathcal{A}(1^\kappa)^{\mathsf{OS\text{-}Ch}^{0'}}, (m, s') \leftarrow S(O, cid)^{\mathsf{OS\text{-}Ch}^{0'}(s)} \\ : m \in M(\kappa) \wedge P_{\ell'} \textit{ is honest} \wedge O \notin Q(s')].$$

Predictably Malleable Onion Secrecy. The fourth property that we require is *predictably malleable onion secrecy*, i.e., for every modification to a ciphertext the challenger is able to compute the resulting changes for the plaintext. This even has to hold for faked plaintexts.

In detail, we define a challenger $\mathsf{OS\text{-}Ch}^0$ that provides, a wrapping, a unwrapping and a send and a destruct oracle. In other words, the challenger provides the same oracles as in the onion routing protocol except that the challenger only provides one single session. We additionally define a faking challenger $\mathsf{OS\text{-}Ch}^1$ that provides the same oracles but fakes all onions for which the attacker does not control the final node.

For $\mathsf{OS\text{-}Ch}^1$, we define the maximal paths that the attacker knows from the circuit. A visible subpath of a circuit $(P_i, k_i, cid_i)_{i=1}^{\ell}$ from an honest onion proxy is a minimal subsequence of corrupted parties $(P_i)_{i=u}^{s}$ of $(P_i)_{i=1}^{\ell}$ such that P_{i-1} is honest and either $s = \ell$ or P_{s+1} is honest as well. The parties P_{i-1} and, if existent, P_{s+1} are called the guards of the visible subpath $(P_i)_{i=u}^{s}$. We store visible subpaths by the first $cid = cid_u$.

(setup, ℓ')
  do the same as OS-Ch^0
  additionally $k_S \leftarrow \{0,1\}^\kappa$

(compromise, i)
  do the same as OS-Ch^0

(send, m)
  $q(st_f^1) \leftarrow m$
  look up the first visible subpath $(cid_1, \langle k_i \rangle_{i=1}^{j})$
  if $j = \ell'$ **then** $m' \leftarrow q(st_f^1)$
  else $k_{j+1} \leftarrow k_S$; $j \leftarrow j+1$; $m' \leftarrow 0^{|q(st_f^1)|}$
  $((O_i)_{i=0}^{j}, s') \leftarrow WrOn^j(m, \langle k_i \rangle_{i=1}^{j}, st_f^1)$
  update $st_f^1 \leftarrow s'$
  store $onions(cid_j) \leftarrow O_1$; send O_j

(unwrap, O, cid_i)
  look up the forward v.s. $\langle k_i \rangle_{i=u}^{j}$ for cid_i
  $O' \leftarrow onions(cid_i)$
  $T \leftarrow \mathrm{D}(O, O')$; $q(st_f^i) \leftarrow T(q(st_f^i))$
  if $j = \ell'$ **then** $m \leftarrow q(st_f^i)$
  else $k_{j+1} \leftarrow k_S$; $j \leftarrow j+1$; $m \leftarrow 0^{|q(st_f^i)|}$
  $((O_i)_{i=u-1}^{j}, s') \leftarrow WrOn^{j-u+1}(m, \langle k_i \rangle_{i=u}^{j}, st_f^i)$
  update $st_f^i \leftarrow s'$
  store $onions(cid_j) \leftarrow O_u$; send O_j

(respond, m)
  $q(st_b^{\ell'}) \leftarrow m$
  look up the last visible subpath $\langle k_i \rangle_{i=u}^{\ell'}$
  if $u = 1$ **then** $m \leftarrow q(st_b^{\ell'})$
  else $k_{u-1} \leftarrow k_S$; $u \leftarrow u-1$; $m \leftarrow 0^{|q(st_b^{\ell'})|}$
  $((O_i)_{i=u-1}^{j}, s') \leftarrow WrOn^{j-u+1}(m, \langle k_i \rangle_{i=u}^{j}, st_b^{\ell'})$
  update $st_b^{\ell'} \leftarrow s'$
  store $onions(cid_u) \leftarrow O_u$; send O_j

(wrap, O, cid_i)
  look up the backward v.s. $\langle k_i \rangle_{i=u}^{j}$ for cid_i
  $O' \leftarrow onions(cid_i)$; $T \leftarrow \mathrm{D}(O, O')$
  $q(st_b^i) \leftarrow T(q(st_b^i))$
  get $\langle k_i \rangle_{i=u}^{j}$ for cid
  if $u = 1$ **then** $m \leftarrow q(st_b^i)$
  else $k_{u-1} \leftarrow k_S$; $u \leftarrow u-1$; $m \leftarrow 0^{|q(st_b^i)|}$
  $((O_i)_{i=u-1}^{j}, s') \leftarrow WrOn^{j-u+1}(m, \langle k_i \rangle_{i=u}^{j}, st_b^i)$
  update $st_b^i \leftarrow s'$
  store $onions(cid_u) \leftarrow O_u$; send O_j

(destruct, O, cid)
  $m \leftarrow UnwrOn(, k_1, st_b^1)$
  $O' \leftarrow onions(cid_1)$; $T \leftarrow \mathrm{D}(O, O')$
  $q(st_b^1) \leftarrow T(q(st_b^1))$
  if $m \neq \bot$ **then**
    send $q(st_b^1)$

Figure 9: The Faking Onion Secrecy Challenger OS-Ch^1: OS-Ch^1 only answers for honest parties. st_f^i, st_b^i is the current forward, respectively backward, state of party i. $((O_i)_{i=u-1}^{j}, s') \leftarrow WrOn^{j-u+1}(m, \langle k_i \rangle_{i=u}^{j}, st)$ is defined as $O_{u-1} \leftarrow m$; **for** i = u **to** j **do** $(O_i, s') \leftarrow WrOn(O_{i-1}, k_{j+u-i}, st)$

In Figure 8 and 9 , we OS-Ch^0, and OS-Ch^1 respectively.[8]

Definition 5 (Predictably malleable onion secrecy)**.** *Let* onionAlg *be a pair of algorithms WrOn and UnwrOn. We say that the algorithms* onionAlg *satisfy* predictably malleable onion secrecy *if there is a negligible function μ such that there is a efficiently computable function* D *such that for all PPT machines $\mathcal{A}$ and sufficiently large κ*

$$\Pr[b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{A}(1^\kappa)^{\mathsf{OS\text{-}Ch}^b} : b = b'] \leq 1/2 + \mu(\kappa)$$

Definition 6 (Secure onion algorithms)**.** *A pair of onion algorithms (WrOn, UnwrOn) is* secure *if is satisfies onion correctness, synchronicity, predictably malleable onion secrecy, and end-to-end integrity.*

In Section 6.2, we show that the Tor algorithms are secure onion algorithms.

4.3 One-Way Authenticated Key-Exchange

We introduce the 1W-AKE primitive in Section 2.2, and later use the 1W-AKE algorithms *Initiate*, *Respond*, and *ComputeKey* in the OR protocol Π_{OR} in Section 2.3. In this section, we give an informal description of the security requirements for 1W-AKE. The rigorous definitions can be found in [13].

The 1W-AKE establishes a symmetric key between two parties (an initiator and a responder) 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 responder cannot be impersonated. *Initiate* takes as input the public key of the responder and generates a challenge. *Respond* takes as input the responder's secret key and the received challenge, and outputs a session key and a response. The algorithm *ComputeKey* runs on the response and the responder's public key, and outputs the session key or an error message of authentication failure.

[8] We stress that in Figure 9 the onion O_u denotes the onion from party P_j to party P_{j+1}.

We assume a public-key infrastructure; i.e., every party knows a secret key whose corresponding public key has been distributed in a verifiable manner. Let pk_P be the public key of party P and sk_P be its secret key.

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 7 (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 $AKE := (Initiate, Respond, ComputeKey)$ 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:*

$$\begin{aligned}\Pr[&(ake, B, m_1, \Psi_A) \leftarrow Initiate(pk_B, B, m),\\ &((ake, B, m_2), (k_2, \star, \vec{v})) \leftarrow Respond(pk_B, sk_B, m_1),\\ &(k_1, B, \vec{v}') \leftarrow ComputeKey(pk_B, m_2,)\\ &: k_1 = k_2 \text{ and } \vec{v} = \vec{v}'] = 1.\end{aligned}$$

The 1W-AKE Challenger for Security. Goldberg, Stebila, and Ustaoglu [13] 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. 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*. This challenger is constructed in a way that security of the 1W-AKE implies one-way authentication of the responding party.

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 $\mathsf{send}^P(m)$.

- $\mathsf{send}^P(params, P')$*:* Compute $(m, st) \leftarrow Initiate\ (pk_P, P', params)$. Send m to the attacker.
- $\mathsf{send}^P(\Psi, msg, P')$*:* If $P' = P$ and $akestate(\Psi) = \bot$, compute $(m, \mathsf{result}) \leftarrow Respond(sk_P, P, msg, \Psi)$. Otherwise, if $msg = (msg', Q)$ compute $(m, \mathsf{result}) \leftarrow ComputeKey(pk_Q, msg', akestate(\Psi), \Psi)$. Then, send m to the attacker.
- $\mathsf{compromise}^P$*:* The challenger returns the long-term key of P to the attacker.

If any verification fails, i.e. one of the algorithms outputs $\bot$, 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 $b \leftarrow \{0, 1\}$. 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 no party involved in that session is compromised.

Definition 8 (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.

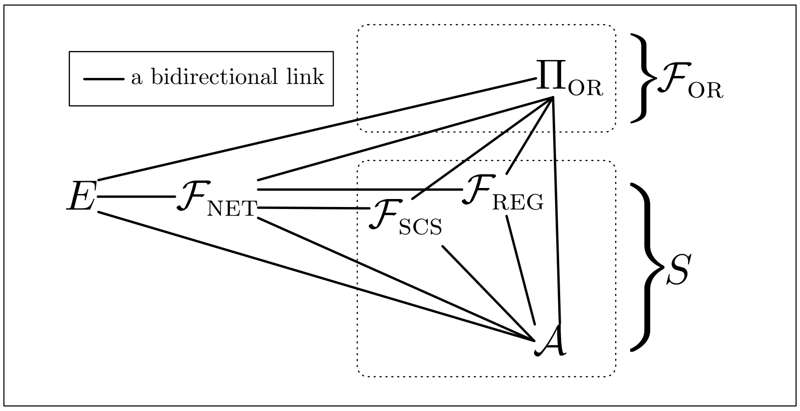

Figure 10: Overview of the set-up

$\mathsf{start}^C(i, j, params, P)$: Abort if $i = j$. Otherwise, set $i \xleftarrow{\$} \{i, j\}$ 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 9 (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 $\mathsf{compromise}^P$ 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 7, 8, and 9.

In Section 6.3 we show that ntor is a 1W-AKE.

5 Π_{OR} UC-Realizes $\mathcal{F}_{\text{OR}}$

In this section, we show that Π_{OR} can be securely abstracted as the ideal functionality $\mathcal{F}_{\text{OR}}$.

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 $\mathcal{F}'$. Recall that $\mathcal{F}_{\text{REG}}$ is the key registration, $\mathcal{F}_{\text{SCS}}$ is the secure channel functionality, and $\mathcal{F}_{\text{NET}^q}$ is the network functionality, where q is the upper bound on the corruptable parties. We prove our result in the $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SCS}}$, $\mathcal{F}_{\text{NET}^q}$-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. The network functionality $\mathcal{F}_{\text{NET}^q}$ is an abstraction of a network that is only partially controlled by an attacker.

Theorem 1. *If Π_{OR} uses secure OR modules $\mathcal{M}$, then the resulting protocol Π_{OR} in the $\mathcal{F}_{\text{REG}}$,$\mathcal{F}_{\text{SCS}}$,$\mathcal{F}_{\text{NET}^q}$-hybrid model securely realizes the ideal functionality $\mathcal{F}_{\text{OR}}$ in the $\mathcal{F}_{\text{NET}^q}$-hybrid model for any q.*

Proof. We have to show that for every PPT attacker $\mathcal{A}$ there is a PPT simulator S such that no PPT environment E can distinguish the interaction with $\mathcal{A}$ and Π_{OR} from the interaction with S and $\mathcal{F}_{\text{OR}}$. Given a PPT attacker $\mathcal{A}$, we construct a simulator S that internally runs $\mathcal{A}$ 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 prove this indistinguishability by examining a sequence of six games and proving their pairwise indistinguishability for the environment E.

Game 1: Game_1 is the original setting in which the environment E interacts with the protocol $\Pi_1 = \Pi_{\text{OR}}$ and the attacker $\mathcal{A}$. Moreover, Π_{OR} and $\mathcal{A}$ have access to a certification authority $\mathcal{F}_{\text{REG}}$ and a secure channel functionality $\mathcal{F}_{\text{SCS}}$, and the network messages of all honest parties are sent via $\mathcal{F}_{\text{NET}^q}$.

Game 2: In Game_2 the simulator S_2 internally runs the attacker $\mathcal{A}$ and the functionalities $\mathcal{F}_{\text{SCS}}$ and $\mathcal{F}_{\text{REG}}$. All messages from these entities are forwarded on the corresponding channels and all messages to these entities are forwarded to the corresponding channels. Since S_2 honestly computes the attacker $\mathcal{A}$, $\mathcal{F}_{\text{SCS}}$, and $\mathcal{F}_{\text{REG}}$, Game_1 and Game_2 are perfectly indistinguishable for the environment E.

Game 3: In the protocol Π_3, we modify the session keys that have been established between two uncompromised parties. All parties are one machine and share some state. Instead of using the established key, Π_2 stores a randomly chosen value in the shared state for each established key k. This random value is used as a session key instead of k.

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_2 and Game_3 are computationally indistinguishable.

Game 4: In Game_4, the onions do not contain the real messages anymore but only the constant zero bitstring. Π_4 maintains a shared datastructure q in which the real messages are stored.

Recall that a visible subpath of a circuit $(P_i, k_i, cid_i)_{i=1}^{\ell}$ from an honest onion proxy is a minimal subsequence of corrupted parties $(P_i)_{i=u}^{s}$ of $(P_i)_{i=1}^{\ell}$ such that P_{i-1} is honest and either $s = \ell$ or P_{s+1} is honest as well. The parties P_{i-1} and, if existent, P_{s+1} are called the guards of the visible subpath $(P_i)_{i=u}^{s}$. In particular, the onion proxy is also a guard. Every circuit can be split into a sequence of visible subpaths and guards. Π_4 stores for every circuit $(P_i, k_i, cid_i)_{i=1}^{\ell}$ such a splitting into visible subpaths and guards. These splittings are updated upon each $\mathsf{compromise}$ command.

Upon receiving a send input or a response from a network, Π_4 stores an input message m in a shared datastructure q as follows. For a the guards P, let be cid_P the circuit id for which P knows the key. Let s the state of the wrapping algorithms of the sender before computing the onion. Then, we store $q(cid_P, s) \leftarrow m$ for each P.

The attacker might be able to corrupt onions such that the contained plaintext is changed. Π_4, however, does not rely on the content of the onions anymore but rather looks up the message in the shared memory. Therefore, Π_4 needs a way to derive the changes to the plaintext due to possible modifications of the ciphertexts. At this point our predictable malleability applies, and we use the algorithm D from the onion secrecy definition for computing the changes in the plaintext. However, for computing the changes in the plaintext, we need to store the onions that the receiving guard has to expect. Hence, Π_4 maintains a shared datastructure *onions* indexed by the *cid* of the receiving guard that stores the expected onions.

Π_4 initially draws some distinguished random key k_S, which is later used for a distinguished last wrapping-layer of the constant zero bitstring. Whenever in Π_3 a guard P that is neither the exit node nor the onion proxy would unwrap an onion O with key k and circuit id cid, P looks up $O' = pending(cid)$. Then, it runs $T \leftarrow S(O, O')$ and replaces the real message $m \leftarrow q(cid, st)$ in the shared memory with $T(m)$, where st is the state of the onion algorithms in the forward direction. Then, P unwraps O with the fake flag, i.e., $(O'', st') \leftarrow UnwrOn(O, k_S, \mathsf{fake}, st)$ instead of $UnwrOn(O, k, st)$. We set the fake flag, because the unwrapping has to skip the integrity check; otherwise a corrupted onion would already in the middle of the circuit be stopped in Π_4. However, instead of forwarding O'', P constructs a new onion either for the attacker or for the next guard as follows. P looks up the adjacent visible subpath $(P_i)_{i=u}^{s}$ in forward direction. If $s = \ell$, then P constructs the onion for the attacker. P reads the real message $m \leftarrow q(cid, st)$ from the shared memory and sends a forward onion O_j for the subcircuit $(P_i, k_i, cid_i)_{i=u}^{\ell}$ that contains the message m and is constructed as follows:

$O_{u-1} \leftarrow m$
for $i = u$ **to** j **do** $(O_i, st') \leftarrow WrOn(O_{i-1}, k_{j+u-i}, st)$

Only then, P updates the forward state $st \leftarrow st'$. Thereafter, P stores $q(cid_{P_{j+1}}, st') \leftarrow O_u$, where $cid_{P_{j+1}}$ is the circuit id of the guard P_{j+1}.

If $s < \ell$, P sends a forward onion for the subcircuit $(P_i, k_i, cid_i)_{i=u}^{s+1}$ that contains $0^{|m|}$ instead of m, where we replace for the last layer k_{s+1} by the distinguished key k_S. Again only then, P updates the forward state $st \leftarrow st'$. Analogously, guards that are onion proxies, i.e., construct an onion in forward direction, also only construct an onion for the attacker or the next guard.

Similar to the forward direction, guards that receive an onion O in backward direction do not wrap it further as in Π_3 but first unwrap O with the fake flag and the distinguished key k_S, i.e., $O' \leftarrow UnwrOn(O, k_S, \mathsf{fake})$. Instead of wrapping O as in Π_3, the guard constructs an onion for the adjacent subpath in backward direction as follows. Since P is a guard for the circuit, also the onion proxy is honest, thus $u > 1$. P looks up the adjacent visible subpath $(P_i)_{i=u}^{s}$ in backward direction. Let $m \leftarrow q(cid, st)$ be the real message stored in the shared memory, cid be the circuit id for which P knows the key and s be the state of the onion algorithms in the backward direction. Then, P sends an onion $(O, st') \leftarrow UnwrOn(0^{|m|}, \langle k_i \rangle_{i=u-1}^{s}, st)$, where $k_{u-1} := k_S$. Thereafter, update the backward state $st \leftarrow st'$.

It might happen that the attacker compromised a node in the middle of the circuit and the exit node. Then the attacker sends a random message to an honest node P. In this case, P would honestly unwrap the message. Since the attacker controls the exit node the broken integrity is not realized. But from that point on the guard P is out of sync, i.e., P has a different unwrapping state than the predecessor guards. Consequently, by the synchronicity of the onion algorithms all future messages that are sent from the onion proxy will be garbage. For guards that are out of sync, we only send randomly chosen messages of appropriate length.

Then, by a hybrid argument it follows that any attacker distinguishing Game_3 from Game_4 can be used for breaking onion secrecy or synchronicity, where the hybrids are indexed by the circuits of honest onion proxies in the order in which the circuits are initiated. Hence, Game_3 and Game_4 are indistinguishable.

Game 5: In this setting the simulator remains unchanged, i.e., $S_5 = S_4$, but the protocol Π_5 in addition internally runs the ideal functionality $\mathcal{F}_{\text{OR}}$. We construct Π_5 such that it does not touch information in the send message, i.e., the message to be sent and the circuit, more than forwarding the send message to $\mathcal{F}_{\text{OR}}$. Instead, Π_5 only uses the messages that it receives from $\mathcal{F}_{\text{OR}}$.

$\mathcal{F}_{\text{OR}}$ outputs for every message from a guard to a visible subpath the entire visible subpath, both in forward an backward direction. If the visible subpath contains the exit node, $\mathcal{F}_{\text{OR}}$ even sends the message. Hence, Π_5 can just compute all onions to the next guard or the attacker in the same way as Π_4.

We also have to cope with the case in which Π_4 modifies the real message in the shared state with the transformation T that D computed from the differences in the expected and the received onion. In this case, Π_5 sends a message $(\mathsf{corrupt}, T, h)$ to $\mathcal{F}_{\text{OR}}$.

Moreover, upon the message $(\mathsf{register}, P)$ the simulator computes a pk for party P and sends a message $(\mathsf{register}, P, pk)$ to the internally emulated functionality $\mathcal{F}_{\text{REG}}$. Upon a response $(\mathsf{registered}, \langle P_j, pk_j \rangle_{j=1}^{v})$ from $\mathcal{F}_{\text{REG}}$, we send $(\mathsf{registered}, \langle P_j \rangle_{j=1}^{v})$

Π_5 behaves like Π_4 except for the key agreement messages, which is computed by the simulator instead of the real party. But by the anonymity of the 1W-AKE primitive, the attacker cannot identify the sender with more than negligible probability. Consequently, Game_4 and Game_5 are indistinguishable.

Game 6: In this setting, we replace the protocol with the ideal functionality; i.e., $\Pi_6 = \mathcal{F}_{\text{OR}}$. The simulator $S := S_6$ in Game_6 additionally computes all network messages exactly as Π_5. As Π_5 did not touch the messages from the environment to the ideal functionality, S can compute Π_5 as well.

The ideal functionality behaves towards the environment exactly as Π_{OR}; consequently, it suffices to show that the network messages are indistinguishable. However, as the simulator S just internally runs Π_5, Game_5 and Game_6 are indistinguishable.

□

As our primitives are proven secure in the random oracle model (ROM), the main theorem uses the ROM.

Theorem 2. *If pseudorandom permutations exist, there are secure OR modules* $(\mathsf{ntor}, \mathsf{onionAlgs})$ *such that the protocol* Π_{OR} *in the* $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SCS}}$, $\mathcal{F}_{\text{NET}^q}$*-hybrid model using* $(\mathsf{ntor}, \mathsf{onionAlgs})$ *securely realizes in the ROM the ideal functionality* $\mathcal{F}_{\text{OR}}$ *in the* $\mathcal{F}_{\text{NET}^q}$*-hybrid model for any* q.

Proof. If pseudorandom permutations exist Lemma 2 implies that secure onion algorithms exist. Lemma 3 shows that in the ROM 1W-AKE exist. Then, Theorem 1 implies the statement. □

Note that we could not prove 1W-AKE security for the TAP protocol currently used in Tor as it uses a CCA-insecure version of the RSA encryption scheme.

6 Instantiating Secure OR Modules

We present a concrete instantiation of OR modules and show that this instantiation constitutes a set of secure OR modules. As onion algorithms we use the algorithms that are used in Tor with a strengthened integrity mechanism, and as 1W-AKE we use the recently proposed ntor protocol [13].

$\mathrm{G}_c(1^\eta)$
  output $k \xleftarrow{\$} \mathrm{G}(1^\eta)$

$\mathrm{E}_c((x_1, \ldots, x_t), (k, ctr)) = \mathrm{D}_c((x_1, \ldots, x_t), (k, ctr))$
  if $ctr = \varepsilon$ **then** $ctr = 0$
  output $(\mathrm{PRP}(s, k) \oplus x_1, \ldots, \mathrm{PRP}(s + t - 1, k) \oplus x_t, (k, ctr + t))$

Figure 11: The stateful deterministic counter-mode (detCTR) $\mathcal{E}_c = (\mathrm{G}_c, \mathrm{E}_c, \mathrm{D}_c)$

We prove that the onion algorithms of Tor constitute secure onion algorithms, as defined in Definition 6. The crucial part in that proof is to show that these onion algorithms are predictably malleable, i.e., for every modification of the ciphertext the changes in the resulting plaintext are predictable by merely comparing the modified ciphertext with the original ciphertext. We first show that the underlying encryption scheme, the deterministic counter-mode, is predictably malleable (Section 6.1). Thereafter, we show the security of Tor's onion algorithms (Section 6.2).

In Section 6.3, we briefly present the ntor protocol and cite the result from Goldberg, Stebila, and Ustaoglu that ntor constitutes a 1W-AKE.

6.1 Deterministic Counter Mode and Predictable Malleability

We show that the deterministic counter-mode (detCTR) scheme is predictably malleable, as defined in Definition 1.

Lemma 1. *If pseudorandom permutations exist, the deterministic counter mode (detCTR) with* $\mathcal{E}_c = (\mathrm{G}_c, \mathrm{E}_c, \mathrm{D}_c)$ *as defined in Figure 11 predictably malleable.*

Proof. We show the result with $t = 1$. This can, however, be extended to larger t in a straight-forward way.

Game_1 is the original game of $\mathcal{A}$ against PM-Ch_1.

Game_2 is the game in which PM-Ch_1 is replaced by the machine B_1 and the PRP Challenger PRP-Ch_1 such that B_1 communicates with $\mathcal{A}$ and PRP-Ch_1, which generates a key applies the PRP candidate algorithms. $\mathcal{A}$ cannot distinguish Game_1 from Game_2, as $\mathcal{A}$'s view is the same in both scenarios.

Game_3 is the game in which PRP-Ch_1 is replaced by PRP-Ch_0, which uses a randomly chosen permutation instead of the PRP candidate. As PRP is a pseudorandom permutation, the attacker cannot distinguish Game_2 from Game_3.

Game_4 is the game in which B_1 is replaced by B_0. Upon a query $(\mathsf{decrypt}, c)$, the B_1 outputs $x \oplus c$ whereas B_0 outputs $c \oplus d \oplus u = c \oplus 0^{|u|} \oplus x \oplus u = c \oplus x \oplus u$. c can be represented as $c = d \oplus c'$ for some bitstring c'. Then, B_1 outputs $x \oplus x \oplus u \oplus c' = u \oplus c'$, and B_0 outputs $x \oplus c' \oplus x \oplus u = u \oplus c'$. Hence, the responses of $(\mathsf{decrypt}, c)$ queries are the same for B_1 and B_0.

Upon a query $(\mathsf{encrypt}, m)$, the B_1 responds $r \oplus m$ whereas B_0 outputs $r \oplus 0^{|m|} = r$. Since, r is randomly chosen and $\oplus$ is a group operation the attacker cannot distinguish $r \oplus m$ from r.[9] Game_3 and Game_4 only differ in B_b; hence, these two games are indistinguishable

Game_5 is the game in which PRP-Ch_0 is replaced by PRP-Ch_1, which uses the PRP candidate instead of a randomly chosen permutation. As PRP is a pseudorandom permutation, the attacker cannot distinguish Game_4 from Game_5.

Game_6 is again the original game of $\mathcal{A}$ against PM-Ch_0. The attacker cannot distinguish Game_5 and Game_6, because the view of $\mathcal{A}$ is the same.

We conclude that Game_1 and Game_6, and therefore PM-Ch_1 and PM-Ch_0, are indistinguishable. □

6.2 Security of Tor's Onion Algorithms

Let $E := (Gen_e, Enc, Dec)$ be a stateful deterministic encryption scheme, and let $M := (Gen_m, Mac, V)$ be a deterministic MAC. Let PRG be a pseudo random generator such that for all $x \in \{0,1\}^*$ $|PRG(x)| =$

[9] Since we use a random permutation, $\mathcal{A}$ can try the following: before starting the *challenge* phase, he sends as many encryption queries as he is allowed to and computes the corresponding $Enc(ctr_e)$. For the *challenge* response c_b he computes $c_b \oplus m$ and checks whether the result equals one of the $Enc(ctr_e)$ he has observed before. If so, either the encryption function is not a permutation or $b = 0$. This, however, only happens with a negligible probability.

upon (initialize)
$ctr_d \leftarrow 0$
$ctr_e \leftarrow 0$

B_b**: upon** (encrypt, m)
send ctr_e to PRP-Ch
wait for input x from PRP-Ch
if $b = 0$ **then**
$c \leftarrow x \oplus 0^{|m|}$
if $q(ctr_e) \neq \perp$ **then**
$(d, u) \leftarrow q(ctr_e)$
$c \leftarrow d$
else if $b = 1$ **then**
$c \leftarrow x \oplus 0^{|m|}$
$q(ctr_e) \leftarrow (c, m)$
increment ctr_e
respond c

B_b**: upon** (decrypt, c)
send ctr_d to PRP-Ch
wait for input x from PRP-Ch
$(d, u) \leftarrow q(ctr_d)$
$T \leftarrow (\lambda y.y \oplus c \oplus d)$
if $b = 0$ **then**
if $q(ctr_d) = \perp$ **then**
$m \leftarrow x \oplus c$
$q(ctr_d) \leftarrow (c, m)$
else if $q(ctr_d) \neq \perp$ **then**
$m \leftarrow T(u)$
else if $b = 1$ **then**
$(m, s) \leftarrow x \oplus c$
$ctr_d \leftarrow s$
respond (m, T)

B_b**: upon** (guess, b^*)
if $b = b^*$ **then**
output 0
else if $b \neq b^*$ **then**
output 1

Figure 12: The machine B_b

$2 \cdot |x|$. We write $PRG(x)_1$ for the first half of $PRG(x)$ and $PRG(x)_2$ the second half. Moreover, for a randomized algorithm A, we write $A(x; r)$ for a call of $A(x)$ with the randomness r.

As a PRP candidate we use AES, as in Tor, and as a MAC use H-MAC with SHA-256. We use that in detCTR encrypting two blocks separately results in the same ciphertext as encrypting the pair of the blocks at once. Moreover, we assume that the output of H-MAC is exactly one block.

The correctness follows by construction. The synchronicity follows, because a PRP is used for the state. The end-to-end integrity directly follows from the SUF of the Mac. And the predictable malleability follows from the predictable malleability of the deterministic counter-mode.

Lemma 2. *Let* onionAlg $= \{UnwrOn_I, WrOn_I\}$*. If pseudorandom permutations exist,* onionAlg *are secure onion algorithms.*

Proof. The correctness follows directly from the construction. The synchronicity can be reduced to the pseudorandomness of PRP of the detCTR scheme. In detail, we can construct a machine that breaks the pseudorandomness of PRP if there is an attacker that breaks the synchronicity.

For showing the end-to-end integrity, assume that there is a ppt attacker that is able to produce an onion O such that $\perp \neq m \leftarrow UnwrOn(O, k)$ and O is not an answer of a query. Then, we can construct a machine that breaks the SUF of the MAC by internally running the attacker against the end-to-end integrity and computing all detCTR call on our own and forwarding all *Mac* calls to the SUF challenger. Finally, after unwrapping the onion, we send the tag t to the SUF challenger as a guess. If the attacker against the end-to-end integrity wins with non-negligible probability, then we also win with non-negligible probability.

For showing the predictable malleability, we present a sequence of games and show that they are indistinguishable for any ppt attacker. In Game_0 the challenger is exactly defined as $\mathsf{OS\text{-}Ch}^0$. In Game_1 additionally the message m is stored in a shared memory $q(st) \leftarrow m$ (st being the corresponding state), and the challenger maintains a separation into visible subpaths. Obviously, Game_1 is indistinguishable from Game_0 for any ppt attacker.

In Game_2, the challenger initially draws a distinguished key k_S. Then, the challenger looks up for every query (unwrap, O, cid_i the adjunct visible subpath $\langle P_i, k_i \rangle_{i=u}^{j}$ in forward direction. Then, the challenger completely unwraps the onion and checks whether the stored message $q(st)$ equals the unwrapped message. If this check fails, the challenger proceeds with the onion as in Game_1. If the this check succeeds, however, and P_j is not the exit node the challenger computes $((O_i)_{i=u}^{j+1}, s') \leftarrow$

$WrOn_I(O,k)$, **for** $m \notin M(\kappa)$
  $O' \leftarrow Enc_{ctr}(O,k)$; return O'

$WrOn_I(m,k)$, **for** $m \in M(\kappa)$
  $(r,r') \leftarrow PRG(k)$; $k_m \leftarrow Gen_m(r)$
  $k_e \leftarrow Gen_e(r')$
  $Mac(m,k_m)$
  $O' \leftarrow Enc_{ctr}(O,k_e)$; return O'

$WrOn_I(m,\langle k_i\rangle_{i=1}^{\ell})$, **for** $m \in M(\kappa)$
  $O_2 \leftarrow WrOn_I(m,k_1)$
  for $i = 2$ to ℓ **do**
    $O_{i+1} \leftarrow WrOn_I(O_i,k_i)$
  return O_ℓ

$UnwrOn_I(O,k)$
  $(r,r') \leftarrow PRG(k)$; $k_m \leftarrow Gen_m(r)$
  $k_e \leftarrow Gen_e(r')$
  $O' \leftarrow Dec_{ctr}(O,k_e)$
  if $O' = m||t$ and $m \in M(\kappa)$ and $V(m,t,k_m) = 1$
  then
    return O''
  else
    $O' \leftarrow Dec_{ctr}(O,k)$; return O'

$UnwrOn_I(O,k,\mathsf{fake})$
  $O' \leftarrow Dec_{ctr}(O,k)$; return O'

$UnwrOn_I(O,\langle k_i\rangle_{i=1}^{\ell})$
  for $i = 1$ to ℓ **do**
    $O_{i+1} \leftarrow WrOn_I(O_i,k_i)$
  return O_ℓ

Figure 13: The Onion Algorithms onionAlg

$WrOn_I^{j-u+2}(m,\langle k_i\rangle_{i=u}^{j+1},st_f^i)$, where $WrOn_I^{j-u+2}$ is defined as in Definition 5, $k_{j+1} := k_S$, and st_f^i denotes the forward state of party i. Thereafter, the challenger updates the forward state $st_f^i \leftarrow s'$ of party i. If P_j is the exit node, then, we only use $\langle k_i\rangle_{i=u}^{j}$, and perform one $WrOn$ operation less. (send,m) queries are processed in the same way except that additionally the message m is stored as $q(st_f^1) \leftarrow m$.

For the backward direction, i.e., queries $(\mathsf{respond},m)$ and (wrap,O,cid_i), the challenger proceeds analogously except that it is not checked whether $P_{\ell'}$ is compromised but whether P_1 is compromised. Accordingly, $k_{u-1} := k_S$ is used in the backward direction. Game_2 is indistinguishable from Game_1 because malicious onions are not touched and the length of a circuit is not leaked by an onion.

In Game_3 the challenger, loosely spoken, fakes all onions for which the message is not visible to the attacker. For all queries, the challenger additionally also stores the onion that the next guard has to expect. For example, consider an onion in forward direction with an adjunct visible subpath $\langle P_i,k_i\rangle_{i=u}^{j}$ for which P_j is not the exit node. Then, the challenger always stores $onions(cid_i) \leftarrow O_u$ the onion that the guard P_{j+1} expects. In the backward direction the challenger analogously stores the expected onion for the next guard. Upon a $(\mathsf{unwrap},O,cid_i)$ query, the challenger runs the predictor $T \leftarrow \mathrm{D}(O,onions(cid_i))$ of detCTR. The resulting transformations T is applied to the stored message $q(st_f^i)$. The challenger proceeds analogously for the query (wrap,O,cid_i) in backward direction. Moreover, the challenger fakes all queries in forward direction for which the last node P_j in the visible subpath $\langle P_i,k_i\rangle_{i=u}^{j}$ is not the exit node, i.e., instead of the actual message $q(st_f^i)$ the constant zero bitstring $0^{|q(st_f^i)|}$ is used.

We can construct a machine M that breaks the predictable malleability of detCTR given an attacker that distinguishes Game_3 from Game_2. M internally runs the attacker computes the challenger Game_3 except for detCTR encryption and decryption calls, which are forwarded to the IND-PM challenger PM-Ch$_i$. Then, M breaks the predictable malleability of detCTR if the attacker distinguishes Game_3 from Game_2.

The challenger in Game_3 is exactly defined as $\mathsf{OS\text{-}Ch}^1$. Since Game_0 and Game_3 are indistinguishable, also $\mathsf{OS\text{-}Ch}^0$ and $\mathsf{OS\text{-}Ch}^1$ are indistinguishable. Hence, onionAlg satisfy predictably malleable onion secrecy. □

6.3 ntor: A 1W-AKE

Øverlier and Syverson [26] proposed a 1W-AKE for use in the next generation of the Tor protocol with improved efficiency. Goldberg, Stebila, and Ustaoglu found an authentication flaw in this proposed protocol, fixed it, and proved the security of the fixed protocol [13]. We use this fixed protocol, called ntor, as a 1W-AKE.

The protocol ntor [13] is a 1W-AKE protocol between two parties P (client) and Q (server), where client P authenticates server Q. Let (pk_Q,sk_Q) be the static key pair for Q. We assume that P holds Q's certificate (Q,pk_Q). P initiates an ntor session by calling the *Initiate* function and sending the output message m_P to Q. Upon receiving a message m'_P, server Q calls the *Respond* function and sends

Initiate(pk_Q, Q):

1. Generate an ephemeral key pair $(x, X \leftarrow g^x)$.
2. Set session id $\Psi_P \leftarrow H_{st}(X)$.
3. Update $st(\Psi_P) \leftarrow (\mathsf{ntor}, Q, x, X)$.
4. Set $m_P \leftarrow (\mathsf{ntor}, Q, X)$.
5. Output m_P.

Respond(pk_Q, sk_Q, X):

1. Verify that $X \in G^*$.
2. Generate an ephemeral key pair $(y, Y \leftarrow g^y)$.
3. Set session id $\Psi_Q \leftarrow H_{st}(Y)$.
4. Compute $(k', k) \leftarrow H(X^y, X^{sk_Q}, Q, X, Y, \mathsf{ntor})$.
5. Compute $t_Q \leftarrow H_{mac}(k', Q, Y, X, \mathsf{ntor}, \mathsf{server})$.
6. Set $m_Q \leftarrow (\mathsf{ntor}, Y, t_Q)$.
7. Set $out \leftarrow (k, \star, X, Y, pk_Q)$, where $\star$ is the anonymous party symbol.
8. Delete y and output m_Q.

ComputeKey(pk_Q, Ψ_P, t_Q, Y):

1. Retrieve Q, x, X from $st(\Psi_P)$ if it exists.
2. Verify that $Y \in G^*$.
3. Compute $(k', k) \leftarrow H(Y^x, pk_Q^x, Q, X, Y, \mathsf{ntor})$.
4. Verify $t_Q = H_{mac}(k', Q, Y, X, \mathsf{ntor}, \mathsf{server})$.
5. Delete $st(\Psi_P)$ and output k.

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

Figure 14: The ntor protocol

the output message m_Q to P. Party P then calls the *ComputeKey* function with parameters from the received message m'_Q, and completes the ntor protocol. We assume a unique mapping between the session ids Ψ_P of the *cid* in Π_{OR}.

Lemma 3 (ntor is anonymous and secure [13])**.** *The* ntor *protocol is a one-way anonymous and secure 1W-AKE protocol in the random oracle model (ROM).*

7 Forward Secrecy and Anonymity Analyses

In this section, we show that our abstraction $\mathcal{F}_{\text{OR}}$ allows for applying previous work on the anonymity analysis of onion routing to Π_{OR}. Moreover, we illustrate that $\mathcal{F}_{\text{OR}}$ allows a rigorous analysis of forward secrecy of Π_{OR}.

In Section 7.1, we show that the analysis of Feigenbaum, Johnson, and Syverson [11] of Tor's anonymity properties in a black-box model can be applied to our protocol Π_{OR}. Feigenbaum, Johnson, and Syverson show their anonymity analysis an ideal functionality $\mathcal{B}_{\text{OR}}$ (see Figure 3). By proving that the analysis of $\mathcal{B}_{\text{OR}}$ applies to $\mathcal{F}_{\text{OR}}$, the UC composition theorem and Theorem 1 implies that the analysis applies to Π_{OR} as well.

In Section 7.2, we prove immediate forward secrecy for Π_{OR} by analyzing $\mathcal{F}_{\text{OR}}$.

7.1 OR Anonymity Analysis

Feigenbaum, Johnson and Syverson [11] analyzed the anonymity properties of OR networks. In their analysis, the authors abstracted an OR network against attackers that are local, static as a black-box functionality $\mathcal{B}_{\text{OR}}$. We reviewed their abstraction $\mathcal{B}_{\text{OR}}$ in Section 2.5. In this section, we show that the analysis of $\mathcal{B}_{\text{OR}}$ is applicable to Π_{OR} against local, static attackers.

There is a slight mismatch in the user-interface of $\mathcal{B}_{\text{OR}}$ and Π_{OR}. The main difference is that Π_{OR} expects separate commands for creating a circuit and sending a message whereas $\mathcal{B}_{\text{OR}}$ only expects a command for sending a message. We construct for every party P a wrapper U for Π_{OR} that adjusts Π_{OR}'s user-interface. Recall that we consider two versions of $\mathcal{B}_{\text{OR}}$ and U simultaneously: one version in which no message is sent and one version in which a message is sent (denoted as $[m]$).

upon the first input m

send N_{OR} to $\mathcal{F}_{\text{REG}}$ in Π

send setup to Π

wait for $(\mathsf{ready}, \langle P_i \rangle_{i=1}^{n})$

further process m

upon an input $(\mathsf{send}, S, [m])$

draw $P_1, \ldots, P_\ell$ at random from N_{OR}

store (S, m_{dummy}) [or (S, m)] in the queue for $\langle P, P_1, \ldots, P_\ell \rangle$

send $(\mathsf{createcircuit}, \langle P, P_1, \ldots, P_\ell \rangle)$ to Π

upon $(\mathsf{created}, \langle P \overset{cid_1}{\Longleftrightarrow} P_1 \Longleftrightarrow \ldots P_\ell \rangle)$ **from** Π

look up (S, m) from the queue for $\langle P, P_1, \ldots, P_\ell \rangle$

send $(\mathsf{send}, \langle P \overset{cid_1}{\Longleftrightarrow} P_1 \Longleftrightarrow \ldots P_\ell \rangle, (S, m))$ to Π

upon $(\mathsf{received}, \mathcal{C}, m)$ **from** Π

do nothing /*$\mathcal{B}_{\text{OR}}$ does not allow responses to messages*/

upon a message m **from** $\mathcal{F}_{\text{NET}^0}$ **to the environment**

forward m to the environment

Figure 15: User-interface $U(\Pi)$ for party P

Instead of Π_{OR}, U only expects one command: $(\mathsf{send}, S, [m])$. We fix the length ℓ of the circuit.[10] Upon $(\mathsf{send}, S, [m])$, $U(\Pi)$ draws the path $P_1, \ldots, P_\ell \overset{\$}{\leftarrow} N_{\text{OR}}$ at random, sends a $(\mathsf{createcircuit}, \langle P, P_1, \ldots, P_\ell \rangle)$ to Π, waits for the *cid* from Π, and sends a (send, cid, m) command, where m is a dummy message if no message is specified. Moreover, in contrast to $\mathcal{B}_{\text{OR}}$ the protocol Π_{OR} allows a response for a message m and therefore additionally sends a session id *sid* to a server.[11]

In addition to the differences in the user-interface, $\mathcal{B}_{\text{OR}}$ assumes the weaker threat model of a local, static attacker whereas Π_{OR} assumes a partially global attacker. We formalize a local attacker by considering Π_{OR} in the $\mathcal{F}_{\text{NET}^0}$-hybrid model, and connect the input/output interface of $\mathcal{F}_{\text{NET}^0}$ to the wrapper U as well. For considering a static attacker, we make the standard UC-assumption that every party only accepts $\mathsf{compromise}$ requests at the beginning of the protocol.

Finally, we also need to assume that $\mathcal{B}_{\text{OR}}$ is defined for a fixed set of onion routers.

Finally, our work culminates in the connection of previous work on black-box anonymity analyses of onion routing with our cryptographic model of onion routing.

Lemma 4 ($U(\Pi_{\text{OR}})$ UC realizes $\mathcal{B}_{\text{OR}}$)**.** *Let $U(\Pi_{\text{OR}})$ be defined as in Figure 15. If Π_{OR} uses secure OR modules, then $U(\Pi_{\text{OR}})$ in the $\mathcal{F}_{\text{NET}^0}$-hybrid model UC realizes $\mathcal{B}_{\text{OR}}$ against static attackers.*

Proof. Applying the UC composition theorem, it suffices to prove that $U(\mathcal{F}_{\text{OR}})$ in the $\mathcal{F}_{\text{NET}^0}$-hybrid model UC realizes $\mathcal{B}_{\text{OR}}$ against static attackers. We construct a simulator $S_{\mathcal{A}}$ as in Figure 16 that internally runs $\mathcal{F}_{\text{NET}^0}$ and $U'(\mathcal{F}_{\text{OR}})$ and an attacker $\mathcal{A}$. Then, we show that $\mathcal{B}_{\text{OR}}$ against $S_{\mathcal{A}}$ is indistinguishable from $U(\mathcal{F}_{\text{OR}})$ against $\mathcal{A}$ for any ppt environment E.

We show that the following sequence of games is indistinguishable for the environment E. The first game Game_1 is the original setting with $U(\mathcal{F}_{\text{OR}})$ and $\mathcal{A}$ in the $\mathcal{F}_{\text{NET}^0}$-hybrid model. In the second game Game_2, the simulator S_2 honestly simulates $\mathcal{F}_{\text{NET}^0}$ and the attacker $\mathcal{A}$. As S_1 honestly simulates $\mathcal{F}_{\text{NET}^0}$ the two games Game_1 and Game_2 are indistinguishable.

In the third game Game_3, the simulator S_3 honestly runs $U(\mathcal{F}_{\text{OR}})$ as well. As S_3 honestly simulates $U(\mathcal{F}_{\text{OR}})$ the two games Game_2 and Game_3 are indistinguishable.

In the fourth game Game_4, the simulator S_4 maintains a set of compromised parties $N_{\mathcal{A}}$. S_4 runs U' instead of U, where U' gets the path as input instead of drawing the path at random. Then, the simulator S_4 upon an input $(\mathsf{send}, S, [m])$ to U draws the first onion router P_1 (not the onion proxy) and the exit node P_ℓ as follows with $b := |N_{\mathcal{A}}|/|N_{\text{OR}}|$.

[10] We fix the length for the sake of brevity. This choice is rather arbitrary. The analysis can be adjusted to the case in which the length is chosen from some efficiently computable distribution or specified by the environment for every message.

[11] It is also possible to modify Π_{OR} such that Π_{OR} does not accept responses and does not draw a session id *sid*. However, for the sake of brevity we slightly modify $\mathcal{B}_{\text{OR}}$.

upon the first input m
set $N_{\mathcal{A}} := \emptyset$
send N_{OR} to $\mathcal{F}_{\text{REG}}$ in $\mathcal{F}_{\text{OR}}$
send setup to $\mathcal{F}_{\text{OR}}$
wait for $(\mathsf{ready}, \langle P_i \rangle_{i=1}^{n})$
further process m

upon $(\mathsf{compromise}, P)$ **from** $\mathcal{A}$
if all previous messages only were $\mathsf{compromise}$ messages **then**
set $N_{\mathcal{A}} := N_{\mathcal{A}} \cup \{P\}$
forward $(\mathsf{compromise}, P)$ to party P in $U(\mathcal{F}_{\text{OR}})$

upon the first message m **that is not** $\mathsf{compromise}$ **from** $\mathcal{A}$
send $(\mathit{compromise}, N_{\mathcal{A}})$ to $\mathcal{B}_{\text{OR}}$
further process m

upon any other message m **from** $\mathcal{A}$ **to** $\mathcal{F}_{\text{NET}^0}$
forward m to $\mathcal{F}_{\text{NET}^0}$

upon any other message m **from** $\mathcal{A}$ **to** $U'(\mathcal{F}_{\text{OR}})$
forward m to $U'(\mathcal{F}_{\text{OR}})$

upon a message m **from** $U'(\mathcal{F}_{\text{OR}})$ **to the environment**
do nothing /* $\mathcal{B}_{\text{OR}}$ already outputs the message */

upon $(\mathsf{sent}, U, S, [m])$ **from** $\mathcal{B}_{\text{OR}}$
choose $P_1 \xleftarrow{\$} \mathsf{N}_{\mathcal{A}}$ and $P_\ell \xleftarrow{\$} \mathsf{N}_{\mathcal{A}}$
choose $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$
send $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m])$ to $U'(\mathcal{F}_{\text{OR}})$

upon $(\mathsf{sent}, -, S, [m])$ **from** $\mathcal{B}_{\text{OR}}$
choose $P_1 \xleftarrow{\$} \mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}}$ and $P_\ell \xleftarrow{\$} \mathsf{N}_{\mathcal{A}}$
choose $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$
send $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m])$ to $U'(\mathcal{F}_{\text{OR}})$

upon $(\mathsf{sent}, U, -)$ **from** $\mathcal{B}_{\text{OR}}$
choose $P_1 \xleftarrow{\$} \mathsf{N}_{\mathcal{A}}$ and $P_\ell \xleftarrow{\$} \mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}}$
choose $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$
send $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m_{\mathsf{dummy}}])$ to $U'(\mathcal{F}_{\text{OR}})$

upon $(\mathsf{sent}, -, -)$ **from** $\mathcal{B}_{\text{OR}}$
choose $P_1 \xleftarrow{\$} \mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}}$ and $P_\ell \xleftarrow{\$} \mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}}$
choose $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$
send $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m_{\mathsf{dummy}}])$ to $U'(\mathcal{F}_{\text{OR}})$

Figure 16: The simulator $S_{\mathcal{A}}$: U' gets the path as input instead of drawing it at random

(i) with probability b^2, S_4 draws $P_1, P_\ell \xleftarrow{\$} N_{\mathcal{A}}$

(ii) with probability $b(1-b)$, S_4 draws $P_1 \xleftarrow{\$} N_{\mathcal{A}}$ and $P_\ell \xleftarrow{\$} N_{\text{OR}} \setminus N_{\mathcal{A}}$

(iii) with probability $(1-b)b$, S_4 draws $P_1 \xleftarrow{\$} N_{\text{OR}} \setminus N_{\mathcal{A}}$ and $P_\ell \xleftarrow{\$} N_{\mathcal{A}}$

(iv) with probability $(1-b)^2$ S_4 draws $P_1, P_\ell \xleftarrow{\$} N_{\text{OR}} \setminus N_{\mathcal{A}}$

The nodes $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$ are drawn uniformly at random. Then, S_4 sends $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m])$ to U'.

Game_4 is indistinguishable from Game_3 as the distribution of compromised parties remains the same, and in Game_4 the modified wrapper U' together with the simulator S_4 have the same input/output behavior as U.

The game Game_5 is the scenario in which $S_{\mathcal{A}}$ communicates with $\mathcal{B}_{\text{OR}}$. The simulator does not directly communicate with the environment over the protocol interface anymore but $\mathcal{B}_{\text{OR}}$ communicates with the environment instead. The simulator S_5 behaves as $S_{\mathcal{A}}$ in Figure 16.

The difference between the input/output behavior of $\mathcal{B}_{\text{OR}}$ and the part of S_4 that communicates with U' is minimal. Only for the cases in which the last onion router is not compromised the message m is not sent to U'. In these cases $S_{\mathcal{A}}$ chooses m_{dummy} as a message. But as the ideal functionality does not reveal any information about m if the last node is not compromised, Game_5 and Game_4 are indistinguishable. □

7.1.1 Generalizing $\mathcal{B}_{\text{OR}}$ to partially global attackers

The result from the previous section can be generalized to an onion routing network against partially global attackers. In order to cope with the partially compromised network, the black-box needs to maintain the amount of compromised links, in addition to the number of compromised parties. In this section, we prove that even for $q > 0$ the onion routing protocol $U(\Pi_{\text{OR}})$ realizes this modified black-box $\mathcal{B}_{\text{OR}'}$, which is defined in Figure 17.

The realization proof goes along the lines of the proof of Lemma 4. However, in order to bound the probability that a link between a user and the first onion router or an exit node and a server is compromised, we need to restrict the number of users and servers. Let m be the amount of users and o be the amount of servers.

Lemma 5 ($U(\Pi_{\text{OR}})$ UC realizes $\mathcal{B}_{\text{OR}'}$)**.** *Let $U(\Pi_{\text{OR}})$ be defined as in Figure 15. If Π_{OR} uses secure OR modules, then $U(\Pi_{\text{OR}})$ in the $\mathcal{F}_{\text{NET}^q}$-hybrid model UC realizes $\mathcal{B}_{\text{OR}}$ against static attackers for any $q \in \{0, \ldots, n\}$, where n is the number of onion routers.*

Proof. Applying the UC composition theorem, it suffices to prove that $U(\mathcal{F}_{\text{OR}})$ in the $\mathcal{F}_{\text{NET}^q}$-hybrid model UC realizes $\mathcal{B}_{\text{OR}'}$ against static attackers. We construct a simulator $S'_{\mathcal{A}}$ as in Figure 18 that internally runs $\mathcal{F}_{\text{NET}^0}$ and $U'(\mathcal{F}_{\text{OR}})$ and an attacker $\mathcal{A}$. Then, we show that $\mathcal{B}_{\text{OR}'}$ against $S'_{\mathcal{A}}$ is indistinguishable from $U(\mathcal{F}_{\text{OR}})$ against $\mathcal{A}$ for any ppt environment E.

We show that the following sequence of games is indistinguishable for the environment E. The first game Game_1 is the original setting with $U(\mathcal{F}_{\text{OR}})$ and $\mathcal{A}$ in the $\mathcal{F}_{\text{NET}^q}$-hybrid model. In the second game Game_2, the simulator S_2 honestly simulates $\mathcal{F}_{\text{NET}^q}$ and the attacker $\mathcal{A}$. As S_1 honestly simulates $\mathcal{F}_{\text{NET}^q}$ the two games Game_1 and Game_2 are indistinguishable.

In the third game Game_3, the simulator S_3 honestly runs $U(\mathcal{F}_{\text{OR}})$ as well. As S_3 honestly simulates $U(\mathcal{F}_{\text{OR}})$ the two games Game_2 and Game_3 are indistinguishable.

In the fourth game Game_4, the simulator S_4 maintains a set of compromised parties $N_{\mathcal{A}}$. S_4 runs U' instead of U, where U' gets the path as input instead of drawing the path at random. Then, the simulator S_4 upon an input $(\mathsf{send}, S, [m])$ to U draws the first onion router P_1 (not the onion proxy) and the exit node P_ℓ as follows with $n := |\mathsf{N}_{\text{OR}}|$, $b \leftarrow \frac{|\mathsf{N}_{\mathcal{A}}|}{n}$, $L'_{\mathcal{A}} := L_{\mathcal{A}} \cap (N_{\text{OR}} \setminus N_{\mathcal{A}})^2$, and $c \leftarrow \frac{|L'_{\mathcal{A}}|}{n(n-1)/2}$:

(*i*) with probability $(b+c)^2$, S_4 draws

$$(P_1, P_\ell) \xleftarrow{\$} (\mathsf{N}_{\mathcal{A}} \cup \{P \mid \exists Q.(P,Q) \in L_{\mathcal{A}}\}) \times (\mathsf{N}_{\mathcal{A}} \cup \{P \mid \exists Q.(P,Q) \in L_{\mathcal{A}}\})$$

(*ii*) with probability $(b+c)(1-(b+c))$, S_4 draws

$$(P_1, P_\ell) \xleftarrow{\$} ((\mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}}) \cap \{P \mid (U,P) \notin L_{\mathcal{A}}\}) \times (\mathsf{N}_{\mathcal{A}} \cup \{P \mid (P,S) \in L_{\mathcal{A}}\})$$

(*iii*) with probability $(1-(b+c))(b+c)$, S_4 draws

$$(P_1, P_\ell) \xleftarrow{\$} (\mathsf{N}_{\mathcal{A}} \cup \{P \mid (U,P) \in L_{\mathcal{A}}\}) \times (\mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}} \cap \{P \mid (P,S) \notin L_{\mathcal{A}}\})$$

(*iv*) with probability $(1-(b+c))^2$, S_4 draws

$$(P_1, P_\ell) \xleftarrow{\$} (\mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}} \cap \{P \mid (U,P) \notin L_{\mathcal{A}}\}) \times (\mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}} \cap \{P \mid (P,S) \notin L_{\mathcal{A}}\})$$

upon receiving a msg ($\mathsf{compromise}, \mathsf{N}_{\mathcal{A}}, L_{\mathcal{A}}$) from $\mathcal{A}$:
set $\mathit{compromised}(P) \leftarrow \mathit{true}$ **for** every $P \in \mathsf{N}_{\mathcal{A}}$
set $n \leftarrow |\mathsf{N}_{\text{OR}}|$; set $b \leftarrow \frac{|\mathsf{N}_{\mathcal{A}}|}{n}$
set $L'_{\mathcal{A}} \leftarrow L_{\mathcal{A}} \cap (\{(P, P') \mid (P \text{ is a user} \wedge P' \in (N_{\text{OR}} \setminus N_{\mathcal{A}})) \vee (P \in (N_{\text{OR}} \setminus N_{\mathcal{A}}) \wedge P' \text{ is a server}\})$
set $c \leftarrow \frac{|L'_{\mathcal{A}}|}{nm+no}$
upon an input ($\mathsf{send}, S, [m]$) from the environment for party U:
with probability $(b+c)^2$,
choose $P_\ell \xleftarrow{\$} \mathsf{N}_{\mathcal{A}}$
send ($\mathsf{sent}, U, S, [m]$) to $\mathcal{A}$
with probability $(1-(b+c))(b+c)$,
choose $P_\ell \xleftarrow{\$} \mathsf{N}_{\mathcal{A}}$
send ($\mathsf{sent}, -, S, [m]$) to $\mathcal{A}$
with probability $(b+c)(1-(b+c))$,
choose $P_\ell \xleftarrow{\$} \mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}}$
send ($\mathsf{sent}, U, -$) to $\mathcal{A}$
with probability $(1-(b+c))^2$,
choose $P_\ell \xleftarrow{\$} \mathsf{N}_{\text{OR}} \setminus \mathsf{N}_{\mathcal{A}}$
send ($\mathsf{sent}, -, -$) to $\mathcal{A}$
output message ($P_\ell, S, [m]$)

Figure 17: Black-box OR Functionality $\mathcal{B}_{\text{OR}'}$ for partially global attackers: N_{OR} is the set of all parties

The nodes $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$ are drawn uniformly at random. Then, S_4 sends ($\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m]$) to U'.

Game_4 is indistinguishable from Game_3 as the distribution of compromised parties remains the same, and in Game_4 the modified wrapper U' together with the simulator S_4 have the same input/output behavior as U.

The game Game_5 is the scenario in which $S'_{\mathcal{A}}$ communicates with $\mathcal{B}_{\text{OR}'}$. The simulator does not directly communicate with the environment over the protocol interface anymore but $\mathcal{B}_{\text{OR}'}$ communicates with the environment instead. The simulator S_5 behaves as $S'_{\mathcal{A}}$ in Figure 18.

The difference between the input/output behavior of $\mathcal{B}_{\text{OR}'}$ and the part of S_4 that communicates with U' is minimal. Only for the cases in which the last onion router is not compromised and the last link is not observed the message m is not sent to U'. In these cases $S'_{\mathcal{A}}$ chooses m_{dummy} as a message. But as the ideal functionality does not reveal any information about m if the last node is not compromised, Game_5 and Game_4 are indistinguishable. □

Extending $\mathcal{B}_{\text{OR}'}$ to reusing circuits. Reusing a circuit, in particular accepting answers, raises the problem that the attacker might learn something by observing activities at the same places. This problem suggests that the resulting abstraction cannot be much simpler than abstraction $\mathcal{F}_{\text{OR}}$.

7.2 Forward Secrecy

Forward secrecy [6] in cryptographic constructions ensures that a session key derived from a set of long-term public and private keys will not be compromised once the session is over, even when one of the (long-term) private keys is compromised in the future. Forward secrecy in onion routing typically refers to the privacy of a user's circuit against an attacker that marches down the circuit compromising the nodes until he reaches the end and breaks the user's anonymity.

It is commonly believed that for achieving forward secrecy in OR protocols it is sufficient to securely erase the local circuit information once a circuit is closed, and to use a key-exchange that provides forward secrecy. Π_{OR} uses such a mechanism for ensuring forward secrecy. Forward secrecy for OR, however, has never been proven, not even rigorously defined.

In this section, we present a game-based definition for OR forward secrecy (Definition 13) and show that Π_{OR} satisfies our forward secrecy definition (Lemma 8). We require that a local attacker does even learn anything about a closed circuit if he compromises all system nodes. The absence of knowledge about a circuit is formalized in the notion of *OR circuit secrecy* (Definition 13), a notion that might be of independent interest.

upon the first input m
set $N_\mathcal{A} := \emptyset$
set $L_\mathcal{A} := \emptyset$
send N_{OR} to $\mathcal{F}_{\text{REG}}$ in $\mathcal{F}_{\text{OR}}$
send setup to $\mathcal{F}_{\text{OR}}$
wait for $(\mathsf{ready}, \langle P_i \rangle_{i=1}^n)$
further process m

upon $(\mathsf{compromise}, P)$ **from** $\mathcal{A}$
if all previous messages were only compromise or obverse messages **then**
set $N_\mathcal{A} := N_\mathcal{A} \cup \{P\}$
forward $(\mathsf{compromise}, P)$ to party P in $U(\mathcal{F}_{\text{OR}})$

upon $(\mathsf{obverse}, P_1, P_2)$ **from** $\mathcal{A}$ **to** $\mathcal{F}_{\text{NET}^q}$
if all previous messages were only compromise or obverse messages **then**
set $L_\mathcal{A} := L_\mathcal{A} \cup \{(P_1, P_2)\}$
forward $(\mathsf{obverse}, P_1, P_2)$ to $\mathcal{F}_{\text{NET}^q}$

upon the first message m **that is not** compromise **from** $\mathcal{A}$
send $(\mathit{compromise}, N_\mathcal{A}, L_\mathcal{A})$ to $\mathcal{B}_{\text{OR}'}$
further process m

upon any other message m **from** $\mathcal{A}$ **to** $\mathcal{F}_{\text{NET}^q}$
forward m to $\mathcal{F}_{\text{NET}^q}$

upon any other message m **from** $\mathcal{A}$ **to** $U'(\mathcal{F}_{\text{OR}})$
forward m to $U'(\mathcal{F}_{\text{OR}})$

upon a message m **from** $U'(\mathcal{F}_{\text{OR}})$ **to the environment**
do nothing /* $\mathcal{B}_{\text{OR}'}$ already outputs the message */

upon $(\mathsf{sent}, U, S, [m])$ **from** $\mathcal{B}_{\text{OR}'}$
choose $(P_1, P_\ell) \xleftarrow{\$} (\mathsf{N}_\mathcal{A} \cup \{P \mid \exists Q.(P,Q) \in L_\mathcal{A}\}) \times (\mathsf{N}_\mathcal{A} \cup \{P \mid \exists Q.(P,Q) \in L_\mathcal{A}\})$
choose $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$
send $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m])$ to $U'(\mathcal{F}_{\text{OR}})$

upon $(\mathsf{sent}, -, S, [m])$ **from** $\mathcal{B}_{\text{OR}'}$
choose $(P_1, P_\ell) \xleftarrow{\$} ((\mathsf{N}_{\text{OR}} \setminus \mathsf{N}_\mathcal{A}) \cap \{P \mid (U,P) \notin L_\mathcal{A}\}) \times (\mathsf{N}_\mathcal{A} \cup \{P \mid (P,S) \in L_\mathcal{A}\})$
choose $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$
send $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m])$ to $U'(\mathcal{F}_{\text{OR}})$

upon $(\mathsf{sent}, U, -)$ **from** $\mathcal{B}_{\text{OR}'}$
choose $(P_1, P_\ell) \xleftarrow{\$} (\mathsf{N}_\mathcal{A} \cup \{P \mid (U,P) \in L_\mathcal{A}\}) \times (\mathsf{N}_{\text{OR}} \setminus \mathsf{N}_\mathcal{A} \cap \{P \mid (P,S) \notin L_\mathcal{A}\})$
choose $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$
send $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m_{\mathsf{dummy}}])$ to $U'(\mathcal{F}_{\text{OR}})$

upon $(\mathsf{sent}, -, -)$ **from** $\mathcal{B}_{\text{OR}'}$
choose $(P_1, P_\ell) \xleftarrow{\$} (\mathsf{N}_{\text{OR}} \setminus \mathsf{N}_\mathcal{A} \cap \{P \mid (U,P) \notin L_\mathcal{A}\}) \times (\mathsf{N}_{\text{OR}} \setminus \mathsf{N}_\mathcal{A} \cap \{P \mid (P,S) \notin L_\mathcal{A}\})$
choose $P_2, \ldots, P_{\ell-1} \xleftarrow{\$} N_{\text{OR}}$
send $(\mathsf{send}, \langle P_i \rangle_{i=1}^{\ell}, [m_{\mathsf{dummy}}])$ to $U'(\mathcal{F}_{\text{OR}})$

Figure 18: The simulator $S'_\mathcal{A}$: U' gets the path as input instead of drawing it at random

Recall that we formalize a local attacker by considering Π_{OR} in the $\mathcal{F}_{\text{NET}^0}$-hybrid model, i.e., the attacker cannot observe the link between any pair of nodes without compromising any of the two nodes.

Definition 10 (Local attackers)**.** *We say that we consider* a protocol Π against local attackers *if we consider* Π *in the* $\mathcal{F}_{\text{NET}^0}$*-hybrid model.*

$\mathsf{CS\text{-}Ch}_b^\Pi$**: upon** (setup) **from** $\mathcal{A}$
if *initial* $= \bot$ **then**
send (setup) to Π
challenge $\leftarrow$ *false*
initial $\leftarrow$ *true*

$\mathsf{CS\text{-}Ch}_b^\Pi$**: upon** (compromise, P) **from** $\mathcal{A}$
if *challenge* $=$ *false* **then**
store that P is compromised
forward (compromise, P) to Π

$\mathsf{CS\text{-}Ch}_b^\Pi$**: upon** (close_initial) **from** $\mathcal{A}$
challenge $\leftarrow$ *true*

$\mathsf{CS\text{-}Ch}_b^\Pi$**: upon** (createcircuit, $\mathcal{P}^0, \mathcal{P}^1, P$) **from** $\mathcal{A}$
if *challenge* $=$ *true* **then**
if $\mathcal{P}^0$ and $\mathcal{P}^1$ visibly coincide **then**
forward (createcircuit, $\mathcal{P}^b, P$) to Π

$\mathsf{CS\text{-}Ch}_b^\Pi$**: for every other message** m **from** $\mathcal{A}$
forward m to Π

$\mathsf{CS\text{-}Ch}_b^\Pi$**: for every message** m **from** Π
if *challenge* $=$ *true* and
$m = (\mathsf{created}, \langle P \overset{cid}{\Longleftrightarrow} P_1 \Longleftrightarrow \cdots P_{\ell'} \rangle, P)$
then
store P for *cid*
forward m to $\mathcal{A}$

OR Circuit Secrecy Challenger: $\mathsf{CS\text{-}Ch}_b^\Pi$

$\mathsf{FS\text{-}Ch}_b^\Pi$ behaves exactly like $\mathsf{CS\text{-}Ch}_b^\Pi$ except for the following message:

$\mathsf{FS\text{-}Ch}_b^\Pi$**: upon** (close_challenge) **from** $\mathcal{A}$
if *challenge* $=$ *true* **then**
challenge $\leftarrow$ *false*
for every circuit *cid* created in the challenge phase **do**
look up onion proxy P for *cid*
send (*cid*, destroy, P) to Π

Figure 19: OR Forward Secrecy Challenger: $\mathsf{FS\text{-}Ch}_b^\Pi$

The definition of circuit secrecy compares a pair of circuits and requires that the attacker cannot tell which one has been used. Of course, we can only compare two circuits the are not trivially distinguishable. The following notion of *visibly coinciding circuits* excludes trivially distinguishable pairs of circuits. Recall that a visible subpath of a circuit is a maximal contiguous subsequence of compromised nodes.

Definition 11 (Visibly coinciding circuits)**.** *A subsequence $\langle P_j \rangle_{j=u}^{s}$ of a circuit $\langle P_i \rangle_{i=1}^{\ell}$ is an* extended visible subpath *if $\langle P_j \rangle_{j=u+1}^{s-1}$ is a visible subpath or $s = \ell$ and $\langle P_j \rangle_{j=u+1}^{s}$ is a visible subpath.*

We say that two circuits $\mathcal{P}^0 = \langle P_i^0 \rangle_{i=0}^{\ell^0}$, $\mathcal{P}^1 = \langle P_i^1 \rangle_{i=0}^{\ell^1}$ are trivially distinguishable if the following three conditions hold:

(i) the onion proxies P_0^0, P_0^1 are not compromised,

(ii) the sequences of extended visible subpaths of $\mathcal{P}^0$ and $\mathcal{P}^1$ are the same, and

(iii) the exit nodes of $\mathcal{P}^0$ and $\mathcal{P}^1$ are the same, i.e., $P_{\ell^0}^0 = P_{\ell^1}^1$.

For the definition of circuit secrecy of a protocol Π, we define a challenger that communicates with the protocol Π and the attacker. The challenger C_b is parametric in a $b \in \{0, 1\}$. C_b forwards all requests from the attacker to the protocol except for the createcircuit commands. Upon a createcircuit command C_b expects a pair $\mathcal{P}^0$, $\mathcal{P}^1$ of node sequences, checks whether $\mathcal{P}^0$ and $\mathcal{P}^1$ are visibly coinciding circuits, chooses $\mathcal{P}^b$, and forwards (createcircuit, $\mathcal{P}^b$) to the protocol Π. We require that the attacker does not learn anything about visibly coinciding circuits.

A protocol can be represented without loss of generality as an interactive Turing machine that internally runs every single protocol party as a submachine, forwards each messages for a party P to that submachine, and sends every message from that submachine to the respective communication partner. We assume that upon a message (setup), a protocol responds with a list of self-generated party identifiers. The protocol expects for every message from the communication partner a party identifier and reroutes the message to the corresponding submachine. In the following definition, we use this notion of a *protocol*.

Definition 12. *Let Π be a protocol and* CS-Ch *be defined as in Figure 19. An OR protocol has circuit secrecy if there is a negligible function μ such that the following holds for all ppt attackers $\mathcal{A}$ and sufficiently large κ*

$$\Pr[b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{A}(\kappa)^{\mathsf{CS\text{-}Ch}_b^\Pi(\kappa)} : b = b'] \leq 1/2 + \mu(\kappa)$$

Forward secrecy requires that even if all nodes are compromised after closing all challenge circuits the attacker cannot learn anything about the challenge circuits.

Definition 13. *Let* Π *be a protocol and* $\mathsf{FS\text{-}Ch}$ *be defined as in Figure 19. An OR protocol has circuit secrecy if there is a negligible function* μ *such that the following holds for all ppt attackers* $\mathcal{A}$ *and sufficiently large* κ

$$\Pr[b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{A}(\kappa)^{\mathsf{FS\text{-}Ch}_b^{\Pi}(\kappa)} : b = b'] \leq 1/2 + \mu(\kappa)$$

Lemma 6. $\mathcal{F}_{\mathrm{OR}}$ *against local attackers satisfies OR circuit secrecy (see Definition 12).*

Proof. As we consider a local attacker the attacker can only observe the communication with compromised nodes, i.e., a guard sends a message to the first compromised node in a visible subpath. For such messages we distinguish two kinds of scenarios: either the visible subpath contains the exit node or not. If the visible subpath contains the exit node, $\mathcal{F}_{\mathrm{OR}}$ sends to the attacker the visible subpath together with the actual message to be transmitted. As any pair of challenge circuits visibly coincides, the visible subpaths are the same; hence, also the messages of $\mathcal{F}_{\mathrm{OR}}$ are the same for $b = 0$ or $b = 1$.

In the case that the visible subpath does not contain the exit node, the circuit contains an adjacent guard on both sides of the visible subpath. In these cases, $\mathcal{F}_{\mathrm{OR}}$ sends the visible subpath, the command relay, and the *cid* to the attacker. As any pair of challenge circuits visibly coincides, the visible subpaths are the same. As the *cid* is randomly chosen, the distributions of the *cid* is the same in the scenario with $b = 0$ and $b = 1$. Consequently, the distribution of network messages is the same in the scenario with $b = 0$ and $b = 1$. □

The protocol as presented in Section 2.3 presents Π_{OR} as one (sub-)machine for every protocol party. Equivalently, Π_{OR} can be represented as one interactive Turing machine that runs all parties as submachines, upon a message (setup) from the communication partner, sends (setup) to every party, and sends an answer with a list of party identifiers to the communication partner. In the following definition, Π_{OR} is represented as one interactive Turing machine that internally runs all protocol parties.

Lemma 7. Π_{OR} *instantiated with secure OR modules against local attackers satisfies OR circuit secrecy (see Definition 12).*

Proof. By Theorem 1, we know that there is a simulator S such that the communication with $\mathsf{CS\text{-}Ch}_b^{\Pi_{\mathrm{OR}}}$ and $\mathsf{CS\text{-}Ch}_b^{\mathcal{F}_{\mathrm{OR}}+S}$ is indistinguishable for any ppt attacker.[12] An attacker $A^{\mathsf{CS\text{-}Ch}_b^{\mathcal{F}_{\mathrm{OR}}+S}}$ communicating with $\mathsf{CS\text{-}Ch}_b^{\mathcal{F}_{\mathrm{OR}}+S}$ can be represented as $S'(A)^{\mathsf{CS\text{-}Ch}_b^{\mathcal{F}_{\mathrm{OR}}}}$ for a wrapping machine S' that upon every network message runs the simulator S and reroutes the network messages of S to the environment to A. By Lemma 6, $S'(A)$ cannot guess b with significantly more than a probability of $1/2$, hence also not $A^{\mathsf{CS\text{-}Ch}_b^{\mathcal{F}_{\mathrm{OR}}+S}}$. As $\mathsf{CS\text{-}Ch}_b^{\Pi_{\mathrm{OR}}}$ and $\mathsf{CS\text{-}Ch}_b^{\mathcal{F}_{\mathrm{OR}}+S}$ are indistinguishable, we conclude that there is no attacker that can guess b with significantly more than a probability of $1/2$.

□

It is easy to see that in $\mathcal{F}_{\mathrm{OR}}$, once a circuit is closed, all information related to the circuit at the uncompromised nodes is deleted. Therefore, forward secrecy for $\mathcal{F}_{\mathrm{OR}}$ is obvious from the circuit secrecy in Lemma 7. Hence, the following lemma immediately follows.

Lemma 8. Π_{OR} *instantiated with secure OR modules against local attackers satisfies OR forward secrecy (see Definition 13).*

8 Conclusions and Future Work

We have proven that the core cryptographic parts in a OR protocol are a one-way anonymous authenticated key exchange primitive (1W-AKE), and secure onion algorithms. We have presented an improved version of the existing Tor protocol using the efficient ntor protocol as a secure 1W-AKE [13] and by proposing provably secure fixes for the Tor onion algorithms with a minimal overhead. We have shown that this improved protocol provides precise security guarantees in a composable setting (UC [4]).

We have further presented an elegant proof technique for the analysis of OR protocols, which leverages an OR abstraction $\mathcal{F}_{\mathrm{OR}}$ that is induced by our UC security result. We show that the analysis of OR

[12] Actually, we do not only consider Π_{OR} but Π_{OR} together with the dummy attacker that only reroutes all messages from the environment to the protocol.

protocol boils down to the analysis of the abstraction $\mathcal{F}_{\text{OR}}$. As an example we have introduced a definition for forward secrecy of onion routing circuits and shown that $\mathcal{F}_{\text{OR}}$ satisfies this definition. Furthermore, we have proven that our abstraction $\mathcal{F}_{\text{OR}}$ satisfies the black-box criteria of Feigenbaum, Johnson and Syverson [11], which in turn implies that their anonymity analysis also applies to the OR protocol presented in this paper.

For future work an interesting direction could be to incorporate hidden services into the UC security analysis. We already designed the abstraction in a way that allows for a modular extension of the UC proof to a hidden service functionality. Moreover, our work offers a framework for the analysis of other desirable OR properties, such as circuit position secrecy.

It is well known that the UC framework lacks a notion of time; consequently any UC security analysis neglects timing attacks, in particular traffic analysis. A composable security analysis that also covers, e.g., traffic analysis, is an interesting task for future work. Although our work proposes a provably secure and practical next generation Tor network, users' anonymity may still be adversely affected if different users run different versions. 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 the Center of IT-Security, Privacy and Accountability (CISPA), NSERC, and Mprime. We thank Gregory Zaverucha for helpful preliminary discussions, and Paul Syverson and Dominique Unruh for their valuable comments on an earlier draft of the paper. We also thank Sebastian Meiser for many insightful discussions about and ideas for predictable malleability.

References

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

[2] V. Boyko, P. D. MacKenzie, and S. Patel, "Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman," in *EUROCRYPT*, 2000, pp. 156–171.

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

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

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

[6] W. Diffie, P. C. van Oorschot, and M. J. Wiener, "Authentication and Authenticated Key Exchanges," *Des. Codes Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.

[7] R. Dingledine and N. Mathewson, "Tor Protocol Specification," https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=tor-spec.txt, 2008, accessed Nov 2011.

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

[9] 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)*, 2007, pp. 57–71.

[10] 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)*, 2007, pp. 1–10.

[11] J. Feigenbaum, A. Johnson, and P. F. Syverson, "Probabilistic Analysis of Onion Routing in a Black-box Model, Tech. Rep. arXiv:1111.2520, 2011, http://arxiv.org/abs/1111.2520v1.

[12] I. Goldberg, "On the Security of the Tor Authentication Protocol," in *Proc. 6th Workshop on Privacy Enhancing Technologies*, 2006, pp. 316–331.

[13] I. Goldberg, D. Stebila, and B. Ustaoglu, "Anonymity and one-way authentication in key exchange protocols," *Designs, Codes and Cryptography*, pp. 1–25, 2012, proposal for Tor: https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/ideas/xxx-ntor-handshake.txt.

[14] O. Goldreich and Y. Lindell, "Session-Key Generation Using Human Passwords Only," in *CRYPTO*, 2001, pp. 408–432.

[15] D. M. Goldschlag, M. Reed, and P. Syverson, "Hiding Routing Information," in *Proc. 1st Workshop on Information Hiding*, 1996, pp. 137–150.

[16] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Onion Routing," *Commun. ACM*, vol. 42, no. 2, pp. 39–41, 1999.

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

[18] D. Hofheinz, "Possibility and Impossibility Results for Selective Decommitments," *J. Cryptology*, vol. 24, no. 3, pp. 470–516, 2011.

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

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

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

[22] A. Kate, G. M. Zaverucha, and I. Goldberg, "Pairing-Based Onion Routing with Improved Forward Secrecy," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 4, p. 29, 2010.

[23] 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)*, 2004, pp. 109–124.

[24] J. McLachlan, A. Tran, N. Hopper, and Y. Kim, "Scalable onion routing with torsk," in *Proc. 16th ACM Conference on Computer and Communication Security (CCS)*, 2009, pp. 590–599.

[25] P. Mittal and N. Borisov, "ShadowWalker: peer-to-peer anonymous communication using redundant structured topologies," in *Proc. 16th ACM Conference on Computer and Communication Security (CCS)*, 2009, pp. 161–172.

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

[27] A. Panchenko, S. Richter, and A. Rache, "NISAN: network information service for anonymization networks," in *Proc. 16th ACM Conference on Computer and Communication Security (CCS)*, 2009, pp. 141–150.

[28] J. Reardon and I. Goldberg, "Improving Tor Using a TCP-over-DTLS Tunnel," in *Proc. 18th USENIX Security Symposium (USENIX)*, 2009, pp. 119–133.

[29] M. Reed, P. Syverson, and D. Goldschlag, "Anonymous Connections and Onion Routing," *IEEE J-SAC*, vol. 16, no. 4, pp. 482–494, 1998.

[30] V. Shmatikov, "Probabilistic analysis of an anonymity system," *Journal of Computer Security*, vol. 12, no. 3-4, pp. 355–377, 2004.

[31] 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)*, 2000, pp. 96–114.

[32] "The Tor Project," https://www.torproject.org/, 2003, accessed Nov 2011.

[33] Y. Zhang, "Effective attacks in the tor authentication protocol," in *NSS '09*, 2009, pp. 81–86.