A Formal Treatment of Onion Routing

Jan Camenisch
IBM Research
Zurich Research Laboratory
CH–8803 Rüschlikon
jca@zurich.ibm.com

Anna Lysyanskaya
Computer Science Department
Brown University
Providence, RI 02912 USA
anna@cs.brown.edu

Abstract. Anonymous channels are necessary for a multitude of privacy-protecting protocols. Onion routing is probably the best known way to achieve anonymity in practice. However, the cryptographic aspects of onion routing have not been sufficiently explored: no satisfactory definitions of security have been given, and existing constructions have only had ad-hoc security analysis for the most part.

We provide a formal definition of onion-routing in the universally composable framework, and also discover a simpler definition (similar to CCA2 security for encryption) that implies security in the UC framework. We then exhibit an efficient and easy to implement construction of an onion routing scheme satisfying this definition.

1 Introduction

The ability to communicate anonymously is requisite for most privacy-preserving interactions. Many cryptographic protocols, and in particular, all the work on group signatures, blind signatures, electronic cash, anonymous credentials, etc., assume anonymous channels as a starting point.

One means to achieve anonymous communication are mix-networks [6]. Here, messages are wrapped in several layers of encryption and then routed through intermediate nodes each of which peels off a layer of encryption and then forwards them in random order to the next node. This process is repeated until all layers are removed. The way messages are wrapped (which determines their path through the network) can either be fixed or can be chosen by each sender for each message.

The former case is usually preferred in applications such as e-voting where one additionally want to ensure that no message is dropped in transit. In that case, each router is required to prove that it behaved correctly: that the messages it outputs are a permutation of the decryption of the messages it has received. The communication model suitable for such a protocol would have a broadcast channel or a public bulletin board; this is not considered efficient in a standard point-to-point network.

In the latter case, where the path is chosen on a message-by-message basis, one often calls the wrapped messages *onions* and speaks of *onion routing* [12, 10]. An onion router is simply responsible for removing a layer of encryption and sending the result to the next onion router. Although this means that onion routing cannot provide robustness (a router may drop an onion and no one will notice), the simplicity of this protocol makes it very attractive in practice. In fact, there are several implementations of onion routing available (see Dingledine et al. [10] and references therein). Unfortunately, these implementations use ad-hoc cryptography instead of provably secure schemes.

The only prior attempt to formalize and construct a provably secure onion routing scheme is due to Möller [16]. Contrary to his claimed goals, it is not hard to see that his definition of security does not guarantee that the onion's distance to destination is hidden from a malicious router. Additionally, his definition does not consider adaptive attacks aimed to break the anonymity properties of onion routing. Thus, although his work represents a first step in the right direction, it falls short of giving a satisfactory definition. His construction does not seem to meet our definition, but has some similarity to our construction.

Alternative means of achieving anonymous communications include Chaum's dining cryptographer networks [7, 8] and Crowds [18].

Onion routing: definitional issues. The state of the literature on anonymous channels today is comparable to that on secure encryption many years ago. While there is a good intuitive understanding of what functionality and security properties an anonymous channel must provide, and a multitude of constructions that seek to meet this intuition, there is a lack of satisfactory *definitions* and, as a result, of provably secure constructions. Indeed, realizing anonymous channels — and constructions aside, simply reasoning about the degree of anonymity a given routing algorithm in a network can provide — remains a question still largely open to rigorous study.

This paper does not actually give a definition of an anonymous channel. We do not know how to define it in such a way that it is, on the one hand, realizable, and, on the other hand, meets our intuitive understanding of what an anonymous channel must accomplish. The stumbling block is that, to realize anonymous channels, one must make non-cryptographic assumptions on the network model. The fact that a solution is proven secure under one set of assumptions on the network does not necessarily imply that it is secure under another set of assumptions.

For example, if one is trying to obtain anonymous channels by constructing a mix network [6], one must make the assumption that (1) there is a dedicated mix network where at least one server is honest; and, more severely, (2) everyone sends and receives about equal amount of traffic and so one cannot match senders to receivers by analyzing the amount of traffic sent and received. In fact, that second assumption on the network was experimentally shown to be crucial — it is known how to break security of mix networks using statistics on network usage where the amount of traffic sent and received by each party is not prescribed to be equal, but rather there is a continuous flow of traffic [14, 9, 23].

In cryptography, however, this is a classical situation. For example, semantic security [13] was introduced to capture what the adversary already knows about the plaintext (before the ciphertext is even formed) by requiring that a cryptosystem be secure for all a-priori distributions on the plaintext, even those chosen by the adversary. Thus, the cryptographic issue of secure encryption, was separated from the non-cryptographic modelling of the adversary's a-priori information. We take a similar approach here.

An onion routing scheme can provide some amount of anonymity when a message is sent through a sufficient number of honest onion routers and there is enough traffic on the network overall. However, nothing can really be inferred about how much anonymity an onion routing algorithm provides without a model that captures network traffic appropriately. As a result, security must be defined with the view of ensuring that the cryptographic aspects of a solution remain secure even in the worst-case network scenario.

Our results. Armed with the definitional approach outlined above, we give a definition of security of an onion routing scheme in the universally composable framework [4]. We chose this approach not because we want onion routing to be universally composable with other protocols (we do, but that's a bonus side effect), but simply because we do not know how to do it in any other way! The beauty and versatility of the UC framework (as well as the related reactive security framework [17, 1]) is that it guarantees that the network issues are orthogonal to the cryptographic ones — i.e., the cryptographic aspects remain secure under the worst-case assumptions on the network behavior. (Similarly to us, Wikström [22] gives a definition of security in the UC framework for general mix networks.)

Definitions based on the UC-framework, however, can be hard to work with. Thus we also give a *cryptographic* definition, similar to CCA2-security for encryption [11]. We show that in order to satisfy our UC-based definition, it is sufficient to give an onion routing scheme satisfying our cryptographic definition.

Finally, we give a construction that satisfies our cryptographic definition.

Overview of our definition and solution. Our ideal functionality does not reveal to an adversarial router any information about onions apart from the prior and the next routers; in particular, the router does not learn how far a given message is from its destination. This property makes traffic analysis a lot harder to carry out, because now any message sent between two onion routers looks the same, even if one of the routers is controlled by the adversary, no matter how close it is to destination [2]. It is actually easy to see where this property comes in. Suppose that it were possible to tell by examining an onion, how far it is from destination. In order to ensure mixing, an onion router that receives an onion O that is h hops away from destination must buffer up several other onions that are also h hops away from destination before sending O to the next router. Overall, if onions can be up to N hops away from destination, each router will be buffering $\Theta(N)$ onions, a few for all possible values of h. This makes onion routing slow and expensive. In contrast, if an onion routing scheme hides distance to destination, then a router may just buffer a constant number of onions before sending them off.

However, achieving this in a cryptographic implementation seems challenging; let us explain why. In onion routing, each onion router P_i, upon receipt of an onion O_i, decrypts it ("peels off" a layer of encryption) to obtain the values

P_{i+1} and O_{i+1}, where P_{i+1} is the identity of the next router in the chain, and O_{i+1} is the data that needs to be sent to P_{i+1}.

Suppose that the outgoing onion O_{i+1} is just the decryption of the incoming onion O_i. Semantic security under the CCA2 attack suggests that, even under active attack from the adversary, if P_i is honest, then the only thing that the incoming onion O_i reveals about the corresponding outgoing onion O_{i+1} is its length.

In the context of encryption, the fact that the length is revealed is a necessary evil that cannot be helped. In this case, however, the problem is not just that the length is revealed, but that, in a secure (i.e., probabilistic) cryptosystem, the length of a plaintext is *always* smaller than the length of a ciphertext.

One attempt to fix this problem is to require that P_i not only decrypt the onion, but also pad it so $|O_i| = |O_{i+1}|$. It is clear that just padding will not work: $|O_{i+1}|$ should be formed in such a way that even P_{i+1} (who can be malicious), upon decrypting O_{i+1} and obtaining the identity of P_{i+2} and the data O_{i+2}, still cannot tell that the onion O_{i+1} was padded, i.e., router P_{i+1} cannot tell that he is not the first router in the chain. At first glance, being able to pad the onion seems to contradict non-malleability: if you can pad it, then, it seems, you can form different onions with the same content and make the scheme vulnerable to adaptive attacks.

Our solution is to use CCA2 encryption *with tags* (or labels) [21, 19, 3], in combination with a pseudorandom permutation (block cipher). We make router P_i pad the onion is such a way that the next router P_{i+1} cannot tell that it was padded; and yet the fact this is possible does not contradict the non-malleability of the scheme because this padding is *deterministic*. The onion will only be processed correctly by P_{i+1} when the tag that P_{i+1} receives is correct, and the only way to make the tag correct is if P_i applied the appropriate deterministic padding. To see how it all fits together, see Section 4.1.

2 Onion Routing in the UC Framework

Setting. Let us assume that there is a network with J players $P_1, \dots, P_J$. For simplicity, we do not distinguish players as senders, routers, and receivers; each player can assume any of these roles. In fact, making such a distinction would not affect our protocol at all and needs to be considered in its application only. We define onion routing in the public key model (i.e., in the hybrid model where a public-key infrastructure is already in place) where each player has an appropriately chosen identity P_i, a registered public key PK_i corresponding to this identity, and these values are known to each player.

In each instance of a message that should be sent, for some (s, r), we have a sender P_s (s stands for "sender") sending a message m of length ℓ_m (the length ℓ_m is a fixed parameter, all messages sent must be the same length) to recipient P_r (r stands for "recipient") through $n < N$ additional routers $P_{o_1}, \dots, P_{o_n}$ (o stands for "onion router"), where the system parameter $N - 1$ is an upper bound on the number of routers that the sender can choose. How each sender selects his onion routers $P_{o_1}, \ldots, P_{o_n}$ is a non-cryptographic problem independent of the current exposition. The input to the onion sending procedure consists of the message m that P_s wishes to send to P_r, a list of onion routers $P_{o_1}, \ldots, P_{o_n}$, and the necessary public keys and parameters. The input to the onion routing procedure consists of an onion O, the routing party's secret key SK, and the necessary public keys and parameters. In case the routing party is in fact the recipient, the routing procedure will output the message m.

Definition of security. The honest players are modelled by imagining that they obtain inputs (i.e., the data m they want to send, the identity of the recipient P_r, and the identities of the onion routers $P_{o_1}, \ldots, P_{o_n}$) from the environment $\mathcal{Z}$, and then follow the protocol (either the ideal or the cryptographic one). Similarly, the honest players' outputs are passed to the environment.

Following the standard universal composability approach (but dropping most of the formalism and subtleties to keep presentation compact), we say that an onion routing protocol is secure if there exists a simulator (ideal-world adversary) $\mathcal{S}$ such that no polynomial-time in λ (the security parameter) environment $\mathcal{Z}$ controlling the inputs and outputs of the honest players, and the behavior of malicious players, can distinguish between interacting with the honest parties in the ideal model through $\mathcal{S}$, or interacting with the honest parties using the protocol.

We note that the solution we present is secure in the public-key model, i.e., in the model where players publish the keys associated with their identities in some reliable manner. In the proof of security, we will allow the simulator $\mathcal{S}$ to generate the keys of all the honest players.

The ideal process. Let us define the ideal onion routing process. Let us assume that the adversary is static, i.e., each player is either honest or corrupted from the beginning, and the trusted party implementing the ideal process knows which parties are honest and which ones are corrupted.

Ideal Onion Routing Functionality: Internal Data Structure.

- The set Bad of parties controlled by the adversary.
- An onion O is stored in the form of $(sid, P_s, P_r, m, n, \mathcal{P}, i)$ where: sid is the identifier, P_s is the sender, P_r is the recipient, m is the message sent through the onion routers, $n < N$ is the length of the onion path, $\mathcal{P} = (P_{o_1}, \ldots, P_{o_n})$ is the path over which the message is sent (by convention, $P_{o_0} = P_s$, and $P_{o_{n+1}} = P_r$), i indicates how much of the path the message has already traversed (initially, $i = 0$). An onion has reached its destination when $i = n + 1$.
- A list L of onions that are being processed by the adversarial routers. Each entry of the list consists of $(temp, O, j)$, where $temp$ is the temporary id that the adversary needs to know to process the onion, while $O = (sid, P_s, P_r, m, n, \mathcal{P}, i)$ is the onion itself, and j is the entry in $\mathcal{P}$ where the onion should be sent next (the adversary does not get to see O and j). Remark: Note that entries are never removed from L. This models the replay attack: the ideal adversary is allowed to resend an onion.

– For each honest party P_i, a buffer B_i of onions that are currently being held by P_i. Each entry consists of (temp', O), where temp' is the temporary id that an honest party needs to know to process the onion and $O = (\mathit{sid}, P_s, P_r, m, n, \mathcal{P}, i)$ is the onion itself (the honest party does not get to see O). Entries from this buffer are removed if an honest party tells the functionality that she wants to send an onion to the next party.

Ideal Onion Routing Functionality: Instructions. The ideal process is activated by a message from router P, from the adversary $\mathcal{S}$, or from itself. There are four types of messages, as follows:

$(\mathsf{Process\_New\_Onion}, P_r, m, n, \mathcal{P})$. Upon receiving such a message from P_s, where $m \in \{0,1\}^{\ell_m} \cup \{\perp\}$, do:

1. If $|\mathcal{P}| \geq N$, reject.
2. Otherwise, create a new session id sid, and let $O = (\mathit{sid}, P, P_r, m, n, \mathcal{P}, 0)$. Send itself message $(\mathsf{Process\_Next\_Step}, O)$.

$(\mathsf{Process\_Next\_Step}, O)$. This is the core of the ideal protocol. Suppose $O = (\mathit{sid}, P_s, P_r, m, n, \mathcal{P}, i)$. The ideal functionality looks at the next part of the path. The router P_{o_i} just processed[1] the onion and now it is being passed to $P_{o_{i+1}}$. Corresponding to which routers are honest, and which ones are adversarial, there are two possibilities for the next part of the path:

I) Honest next. Suppose that the next node, $P_{o_{i+1}}$, is honest. Here, the ideal functionality makes up a random temporary id temp for this onion and sends to $\mathcal{S}$ (recall that $\mathcal{S}$ controls the network so it decides which messages get delivered): "Onion temp from P_{o_i} to $P_{o_{i+1}}$." It adds the entry $(\mathit{temp}, O, i+1)$ to list L. (See $(\mathsf{Deliver\_Message}, \mathit{temp})$ for what happens next.)

II) Adversary next. Suppose that $P_{o_{i+1}}$ is adversarial. Then there are two cases:

– There is an honest router remaining on the path to the recipient. Let P_{o_j} be the next honest router. (I.e., $j > i$ is the smallest integer such that P_{o_j} is honest.) In this case, the ideal functionality creates a random temporary id temp for this onion, and sends the message "Onion temp from P_{o_i}, routed through $(P_{o_{i+1}}, \ldots, P_{o_{j-1}})$ to P_{o_j}" to the ideal adversary $\mathcal{S}$, and stores (temp, O, j) on the list L.

– P_{o_i} is the last honest router on the path; in particular, this means that P_r is adversarial as well. In that case, the ideal functionality sends the message "Onion from P_{o_i} with message m for P_r routed through $(P_{o_{i+1}}, \ldots, P_{o_n})$" to the adversary $\mathcal{S}$. (Note that if $P_{o_i+1} = P_r$, the list $(P_{o_{i+1}}, \ldots, P_{o_n})$ will be empty.)

$(\mathsf{Deliver\_Message}, \mathit{temp})$. This is a message that $\mathcal{S}$ sends to the ideal process to notify it that it agrees that the onion with temporary id temp should be delivered to its current destination. To process this message, the functionality checks if the temporary identifier temp corresponds to any onion O on the list

[1] In case $i = 0$, processed means having originated the onion and submitted it to the ideal process.

L. If it does, it retrieves the corresponding record $(temp, O, j)$ and update the onion: if $O = (sid, P_s, P_r, m, n, \mathcal{P}, i)$, it replaces i with j to indicate that we have reached the j'th router on the path of this onion. If $j < n+1$, it generates a temporary identifier $temp'$, sends "Onion $temp'$ received" to party P_{o_j}, and stores the resulting pair $(temp', O = (sid, P_s, P_r, m, n, \mathcal{P}, j))$ in the buffer B_{o_j} of party P_{o_j}. Otherwise, $j = n+1$, so the onion has reached its destination: if $m \neq \perp$ it sends "Message m received" to router P_r; otherwise it does not deliver anything[2].

($\mathsf{Forward\_Onion}$, $temp'$). This is a message from an honest ideal router P_i notifying the ideal process that it is ready to send the onion with id $temp'$ to the next hop. In response, the ideal functionality

- Checks if the temporary identifier $temp'$ corresponds to any entry in B_i. If it does, it retrieves the corresponding record $(temp', O)$.
- Sends itself the message $(\mathsf{Process\_Next\_Step}, O)$.
- Removes $(temp', O)$ from B_i.

This concludes the description of the ideal functionality. We must now explain how the ideal honest routers work. When an honest router receives a message of the form "Onion $temp'$ received" from the ideal functionality, it notifies environment $\mathcal{Z}$ about it and awaits instructions for when to forward the onion $temp'$ to its next destination. When instructed by $\mathcal{Z}$, it sends the message "$\mathsf{Forward\_Onion}$ $temp'$" to the ideal functionality.

It's not hard to see that $\mathcal{Z}$ learns nothing else than pieces of paths of onions formed by honest senders (i.e., does not learn a sub-path's position or relations among different sub-paths). Moreover, if the sender and the receiver are both honest, the adversary does not learn the message.

2.1 Remarks and Extensions

Mixing strategy. It may seem that, as defined in our ideal functionality, the adversary is too powerful because, for example, it is allowed to route just one onion at a time, and so can trace its entire route. In an onion routing implementation however, the instructions for which onion to send on will not come directly from the adversary, but rather from an honest player's mixing strategy. That is, each (honest) router is notified that an onion has arrived and is given a handle *temp* to that onion. Whenever the router decides (under her mixing strategy) that the onion *temp* should be sent on, she can notify the ideal functionality of this using the handle *temp*. A good mixing strategy will limit the power of the adversary to trace onions in the ideal world, which will translate into limited capability in the real world as well. What mixing strategy is a good one depends on the network. Additionally, there is a trade-off between providing more anonymity and minimizing latency of the network. We do not consider any

[2] This is needed to account for the fact that the adversary inserts onions into the network that at some point do not decrypt correctly.

of these issues in this paper but only point out that our scheme guarantees the maximum degree of security that any mixing strategy can inherently provide.

Replay attacks. The definition as is allows replay attacks by the adversary. The adversary controls the network and can replay any message it wishes. In particular, it can take an onion that party P_i wants to send to P_j and deliver it to P_j as many times as it wishes. However, it is straightforward to modify our security definition and our scheme so as to prevent replay attacks. For instance, we could require that the sender inserts time stamps into all onions. I.e., a router P_i, in addition to the identity of the next router P_{i+1}, will also be given a time $\mathtt{time}$ and a random identifier $\mathtt{oid}_i$ (different for each onion and router). An onion router will drop the incoming onion when either the time $\mathtt{time} + t_\Delta$ (where t_Δ is a parameter) has passed or it finds $\mathtt{oid}_i$ in its database. If an onion is not dropped, the router will store $\mathtt{oid}_i$ until time $\mathtt{time} + t_\Delta$ has passed. It is not difficult to adapt our scheme and model to reflect this. We omit details to keep this exposition focused.

Forward security. Forward secrecy is a desirable property in general, and in this context in particular [5, 10]. Our scheme can be constructed from any CCA2-secure cryptosystem, and in particular, from a forward-secure one.

The response option. Another desirable property of an onion routing scheme is being able to respond to a message received anonymously. We address this after presenting our construction.

3 A Cryptographic Definition of Onion Routing

Here we give a cryptographic definition of an onion routing scheme and show why a scheme satisfying this definition is sufficient to realize the onion routing functionality described in the previous section.

Definition 1 (Onion routing scheme I/O). *A set of algorithms* (G, $\mathsf{FormOnion}$, $\mathsf{ProcOnion}$) *satisfies the I/O spec for an onion routing scheme for message space* $M(1^\lambda)$ *and set of router names* $\mathcal{Q}$ *if:*

- G *is a key generation algorithm, possibly taking as input some public parameters* p*, and a router name* P*:* $(PK, SK) \leftarrow G(1^\lambda, p, P)$.
- $\mathsf{FormOnion}$ *is a probabilistic algorithm that on input a message* $m \in M(1^\lambda)$*, an upper bound on the number of layers* N*, a set of router names* $(P_1, \ldots, P_{n+1})$ *(each* $P_i \in \mathcal{Q}$*,* $n \leq N$*), and a set of public keys corresponding to these routers* $(PK_1, \ldots, PK_{n+1})$*, outputs a set of onion layers* $(O_1, \ldots, O_{n+1})$*. (As* N *is typically a system-wide parameter, we usually omit to give it as input to this algorithm.)*
- $\mathsf{ProcOnion}$ *is a deterministic algorithm that, on input an onion* O*, identity* P*, and a secret key* SK*, peels off a layer of the onion to obtain a new onion* O' *and a destination* P' *for where to send it:* $(O', P') \leftarrow \mathsf{ProcOnion}(SK, O, P)$.

Definition 2 (Onion evolution, path, and layering). *Let* (G, $\mathsf{FormOnion}$, $\mathsf{ProcOnion}$) *satisfy the onion routing I/O spec. Let* p *be the public parame-*

ters. Suppose that we have a set $\mathcal{Q}$, $\perp \notin \mathcal{Q}$, consisting of a polynomial number of (honest) router names. Suppose that we have a public-key infrastructure on $\mathcal{Q}$, i.e., corresponding to each name $P \in \mathcal{Q}$ there exists a key pair $(PK(P), SK(P))$, generated by running $G(1^\lambda, p, P)$. Let O be an onion received by router $P \in \mathcal{Q}$. Let $\mathcal{E}(O,P) = \{(O_i, P_i) \ : \ i \geq 1\}$ be the maximal ordered *list of pairs such that $P_1 = P$, $O_1 = O$, and for all $i > 1$, $P_i \in \mathcal{Q}$, and $(O_i, P_i) = \mathsf{ProcOnion}(SK(P_{i-1}), O_{i-1}, P_{i-1})$. Then $\mathcal{E}(O,P)$ is the* evolution *of onion O starting at P. Moreover, if $\mathcal{E}(O,P) = \{(O_i, P_i)\}$ is the evolution of an onion, then $\mathcal{P}(O,P) = \{P_i\}$ is the* path *of the onion, while $\mathcal{L}(O,P) = \{O_i\}$ is the* layering *of the onion.*

Onion-correctness is the simple condition that if an onion is formed correctly and then the correct routers process it in the correct order, then the correct message is received by the last router P_{n+1}.

Definition 3 (Onion-correctness). *Let* $(G$, $\mathsf{FormOnion}$, $\mathsf{ProcOnion})$ *satisfy the I/O spec for an onion routing scheme. Then for all settings of the public parameters p, for all $n < N$, and for all $\mathcal{Q}$ with a public-key infrastructure as in Definition 2, for any path $\mathcal{P} = (P_1, \ldots, P_{n+1})$,$\mathcal{P} \subseteq \mathcal{Q}$, for all messages $m \in M(1^\lambda)$, and for all onions O_1 formed as*

$$(O_1, \ldots, O_{n+1}) \leftarrow \mathsf{FormOnion}(m, N, (P_1, \ldots, P_{n+1}), (PK(P_1), \ldots, PK(P_{n+1})))$$

the following is true: (1) correct path*: $\mathcal{P}(O_1, P_1) = (P_1, \ldots, P_{n+1})$; (2)* correct layering*: $\mathcal{L}(O_1, P_1) = (O_1, \ldots, O_{n+1})$; (3)* correct decryption*: $(m, \perp) = \mathsf{ProcOnion}(SK(P_{n+1}), O_{n+1}, P_{n+1})$.*

Onion-integrity requires that even for an onion created by an adversary, the path is going to be of length at most N.

Definition 4 (Onion-integrity). *(Sketch) An onion routing scheme satisfies onion-integrity if for all probabilistic polynomial-time adversaries, the probability (taken over the choice of the public parameters p, the set of honest router names $\mathcal{Q}$ and the corresponding PKI as in Definition 2) that an adversary with adaptive access to $\mathsf{ProcOnion}(SK(P), \cdot, P)$ procedures for all $P \in \mathcal{Q}$, can produce and send to a router $P_1 \in \mathcal{Q}$ an onion O_1 such that $|\mathcal{P}(O_1, P_1)| > N$, is negligible.*

Our definition of onion security is somewhat less intuitive. Here, an adversary is launching an adaptive attack against an onion router P. It gets to send onions to this router, and see how the router reacts, i.e., obtain the output of $\mathsf{ProcOnion}(SK(P), \cdot, P)$. The adversary's goal is to distinguish whether a given challenge onion corresponds to a particular message and route, or a random message and null route. The unintuitive part is that the adversary can also succeed by *re-wrapping* an onion, i.e., by adding a layer to its challenge onion.

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

1. *The adversary receives as input a challenge public key PK, chosen by the challenger by letting $(PK, SK) \leftarrow G(1^\lambda, p)$, and the router name P.*
2. *The adversary may submit any number of onions O_i of his choice to the challenger, and obtain the output of* $\mathsf{ProcOnion}(SK, O_i, P)$.
3. *The adversary submits n, a message m, a set of names $(P_1, \ldots, P_{n+1})$, and index j, and n key pairs $1 \leq i \leq n+1$, $i \neq j$, (PK_i, SK_i). The challenger checks that the router names are valid*[3]*, that the public keys correspond to the secret keys, and if so, sets $PK_j = PK$, sets bit b at random, and does the following:*
   - *If $b = 0$, let*

$$(O_1, \ldots, O_j, \ldots, O_{n+1}) \leftarrow \mathsf{FormOnion}(m, (P_1, \ldots, P_{n+1}), (PK_1, \ldots, PK_{n+1}))$$

   - *Otherwise, choose $r \leftarrow M(1^\lambda)$, and let*

$$(O_1, \ldots, O_j) \leftarrow \mathsf{FormOnion}(r, (P_1, \ldots, P_j), (PK_1, \ldots, PK_j))$$

4. *Now the adversary is allowed get responses for two types of queries:*
   - *Submit any onion $O_i \neq O_j$ of his choice and obtain* $\mathsf{ProcOnion}(SK, O_i, P)$.
   - *Submit a secret key SK', an identity $P' \neq P_{j-1}$, and an onion O' such that $O_j = \mathsf{ProcOnion}(SK', O', P')$; if P' is valid, and (SK', O', P') satisfy this condition, then the challenger responds by revealing the bit b.*
5. *The adversary then produces a guess b'.*

We say that a scheme with onion routing I/O satisfies onion security if for all probabilistic polynomial time adversaries $\mathcal{A}$ of the form described above, there is a negligible function ν such that the adversary's probability of outputting $b' = b$ is at most $1/2 + \nu(\lambda)$.

This definition of security is simple enough, much simpler than the UC-based definition described in the previous section. Yet, it turns out to be sufficient. Let us give an intuitive explanation why. A simulator that translates between a real-life adversary and an ideal functionality is responsible for two things: (1) creating some fake traffic in the real world that accounts for everything that happens in the ideal world; and (2) translating the adversary's actions in the real world into instructions for the ideal functionality.

In particular, in its capacity (1), the simulator will sometimes receive a message from the ideal functionality telling it that an onion *temp* for honest router P_j is routed through adversarial routers $(P_1, \ldots, P_{j-1})$. The simulator is going to need to make up an onion O_1 to send to the adversarial party P_1. But the simulator is not going to know the message contained in the onion, or the rest of the route. So the simulator will instead make up a random message r and compute the onion so that it decrypts to r when it reaches the honest (real) router P_j. I.e, it will form O_1 by obtaining

[3] In our construction, router names are formed in a special way, hence this step is necessary for our construction to satisfy this definition.

$(O_1, \ldots, O_j) \leftarrow \mathsf{FormOnion}(r, (P_1, \ldots, P_j), (PK_1, \ldots, PK_j))$. When the onion O_j arrives at P_j from the adversary, the simulator knows that it is time to tell the ideal functionality to deliver message *temp* to honest ideal P_j.

Now, there is a danger that this may cause errors in the simulation as far as capacity (2) is concerned: the adversary may manage to form another onion $\tilde{O}$, and send it to an honest router $\tilde{P}$, such that $(O_j, P) \in \mathcal{E}(\tilde{O}, \tilde{P})$. The simulator will be unable to handle this situation correctly, as the simulator relies on its ability to correctly decrypt and route all real-world onions, while in this case, the simulator does not know how to decrypt and route this "fake" onion past honest router P_j. A scheme satisfying the definition above would prevent this from happening: the adversary will not be able to form an onion $O' \neq O_{j-1}$ sent to an honest player P' such that $(P_j, O_j) = \mathsf{ProcOnion}(SK(P'), O', P')$.

In the full version of this paper, we give a formal proof of the following theorem:

Theorem 1. *An onion routing scheme* $(G, \mathsf{FormOnion}, \mathsf{ProcOnion})$ *satisfying onion-correctness, integrity and security, when combined with secure point-to-point channels, yields a UC-secure onion routing scheme.*

4 Onion Routing Scheme Construction

Tagged encryption. The main tool in our construction is a CCA2-secure cryptosystem (Gen, E, D) that supports tags. Tags were introduced by Shoup and Gennaro [21]. The meaning of a tagged ciphertext is that the tag provides the context within which the ciphertext is to be decrypted. The point is that an adversary cannot attack the system by making the honest party under attack decrypt this ciphertext out of context. The input to E is (PK, m, T), where T is a tag, such that $D(SK, c, T')$ should fail if $c \leftarrow E(PK, m, T)$ and $T' \neq T$. In the definition of CCA2-security for tagged encryption, the adversary is, as usual, given adaptive access to the decryption oracle D throughout its attack; it chooses two messages (m_0, m_1) and a tag T and is given a challenge ciphertext $c \leftarrow E(PK, m_b, T)$ for a random bit b. The adversary is allowed to issue further queries $(c', T') \neq (c, T)$ to D. The definition of security stipulates that the adversary cannot guess b with probability non-negligibly higher than $1/2$. We omit the formal definition of CCA2-security with tags, and refer the reader to prior work.

Pseudorandom permutations. We also use pseudorandom permutations (PRPs). Recall [15] that a polynomial-time algorithm $p_{(\cdot)}(\cdot)$ defines a pseudorandom permutation family if for every key $K \in \{0,1\}^*$, $p_K : \{0,1\}^{\ell(|K|)} \mapsto \{0,1\}^{\ell(|K|)}$ (where the function $\ell(\cdot)$ is upper-bounded by a polynomial, and is called the "block length" of p) is a permutation and is indistinguishable from a random permutation by any probabilistic poly-time adversary $\mathcal{A}$ with adaptive access to both p_K and p_K^{-1}. We have the same key K define a set of simultaneously pseudorandom permutations $\{p_K^i : 1 \leq i \leq \ell(|K|)\}$, where i is the block length for a permutation p_K^i. (This can be obtained from any standard pseudorandom permutation family by standard techniques. For example, let $K_i = F_K(i)$, where F is a pseudorandom function, and let $p^i_K = p^i_{K_i}$.)

Notation. In the sequel, we will denote p^i_K by p_K because the block length is always clear from the context. Let $\{m\}_K$ denote $p_K^{|m|}(m)$. Let $\{m\}_{K^{-1}}$ denote $(p^{-1})_K^{|m|}(m)$. By '$\circ$' we denote concatenation.

Parameters. Let λ be the security parameter. It guides the choice of ℓ_K which is the length of a PRP key, and of ℓ_C, which is the upper bound on the length of a ciphertext formed using the CCA2 secure cryptosystem (Gen, E, D) when the security parameter is λ. Let ℓ_m be the length of a message being sent. Let $\ell_H = \ell_K + \ell_C$.

Non-standard assumption on the PRP. We assume that, if P_1 and P_2 are two strings of length $2\ell_K$ chosen uniformly at random, then it is hard to find N keys $K_1, \ldots, K_N$ and a string C of length ℓ_C such that

$$\{\{\ldots\{P_1 \circ 0^{\ell_C}\}_{K_1^{-1}}\ldots\}_{K_{N-1}^{-1}}\}_{K_N^{-1}} \in \{P_1 \circ C, P_2 \circ C\}$$

In the random-oracle model, it is easy to construct a PRP with this property: if p is a PRP, define p' as $p'_K = p_{\mathcal{H}(K)}$ where $\mathcal{H}$ is a random oracle. If this assumption can hold in the standard model, then our construction is secure in the plain model as well.

4.1 Construction of Onions

We begin with intuition for our construction. Suppose that the sender P_s would like to route a message m to recipient $P_r = P_{n+1}$ through intermediate routers $(P_1, \ldots, P_n)$. For a moment, imagine that the sender P_s has already established a common one-time secret key K_i with each router P_i, $1 \leq i \leq n+1$. In that setting, the following construction would work and guarantee some (although not the appropriate amount of) security:

Intuition: Construction 1. For simplicity, let $N = 4$, $n = 3$, so the sender is sending message m to P_4 via intermediate routers P_1, P_2 and P_3. Send to P_1 the onion O_1 formed as follows:

$$O_1 = (\{\{\{\{m\}_{K_4}\}_{K_3}\}_{K_2}\}_{K_1}, \{\{\{P_4\}_{K_3}\}_{K_2}\}_{K_1}, \{\{P_3\}_{K_2}\}_{K_1}, \{P_2\}_{K_1})$$

Upon receipt of this $O_1 = (M^{(1)}, H_3^{(1)}, H_2^{(1)}, H_1^{(1)})$, P_1 will remove a layer of encryption using key K_1, and obtain

$$\begin{aligned}(\{M^{(1)}\}_{K_1^{-1}}, \{H_3^{(1)}\}_{K_1^{-1}}, \{H_2^{(1)}\}_{K_1^{-1}}, \{H_1^{(1)}\}_{K_1^{-1}}) = \\ (\{\{\{m\}_{K_4}\}_{K_3}\}_{K_2}, \{\{P_4\}_{K_3}\}_{K_2}, \{P_3\}_{K_2}\}, P_2)\end{aligned}$$

Now P_1 knows that P_2 is the next router. It could, therefore, send to P_2 the set of values $(\{M^{(1)}\}_{K_1^{-1}}, \{H_3^{(1)}\}_{K_1^{-1}}, \{H_2^{(1)}\}_{K_1^{-1}})$. But then the resulting onion O_2 will be shorter than O_1, which in this case would make it obvious to P_2 that he is only two hops from the recipient; while we want P_2 to think that he could be up to $N-1$ hops away from the recipient. Thus, P_1 needs to pad the onion somehow. For example, P_1 picks a random string R_1 of length $|P_1|$ and sets:

$$
\begin{aligned}
(O_2, P_2) &= \mathsf{ProcOnion}(K_1, O_1, P_1) \\
&= ((\{M^{(1)}\}_{K_1^{-1}}, R_1, \{H_3^{(1)}\}_{K_1^{-1}}, \{H_2^{(1)}\}_{K_1^{-1}}), \{H_1^{(1)}\}_{K_1^{-1}}) \\
&= ((\{\{\{m\}_{K_4}\}_{K_3}\}_{K_2}, R_1, \{\{P_4\}_{K_3}\}_{K_2}, \{P_3\}_{K_2}\}), P_2)
\end{aligned}
$$

Upon receipt of this $O_2 = (M^{(2)}, H_3^{(2)}, H_2^{(2)}, H_1^{(2)})$, P_2 will execute the same procedure as P_1, but using his key K_2, and will obtain onion O_3 and the identity of router P_3. Upon receipt of O_3, P_3 will also apply the same procedure and obtain O_4 and the identity of the router P_4. Finally, P_4 will obtain:

$$
\begin{aligned}
(O_5, P_5) &= \mathsf{ProcOnion}(K_4, O_4, P_4) \\
&= ((\{M^{(4)}\}_{K_4^{-1}}, R_4, \{H_3^{(4)}\}_{K_4^{-1}}, \{H_2^{(4)}\}_{K_4^{-1}}), \{H_1^{(4)}\}_{K_4^{-1}}) \\
&= ((m, R_4, \{R_3\}_{K_4^{-1}}, \{\{R_2\}_{K_3^{-1}}\}_{K_4^{-1}}), \{\{\{R_1\}_{K_2^{-1}}\}_{K_3^{-1}}\}_{K_4^{-1}})
\end{aligned}
$$

How does P_4 know that he is the recipient? The probability over the choice of K_4 that P_5 obtained this way corresponds to a legal router name is negligible. Alternatively, P_4 may be able to tell if, by convention, a legal message m must begin with k 0's, where k is a security parameter.

Intuition: Construction 2. Let us now adapt Construction 1 to the public-key setting. It is clear that the symmetric keys K_i, $1 \leq i \leq n+1$, need to be communicated to routers P_i using public-key encryption. In Construction 1, the only header information $H_1^{(i)}$ for router P_i was the identity of the next router, P_{i+1}. Now, the header information for router P_i must also include a public-key ciphertext $C_{i+1} = E(PK_{i+1}, K_{i+1}, T_{i+1})$, which will allow router P_{i+1} to obtain his symmetric key K_{i+1}. We need to explain how these ciphertexts are formed. Let us first consider C_1. Tag T_1 is used to provide the context within which router P_1 should decrypt C_1. C_1 exists in the context of the message part and the header of the onion, and therefore the intuitive thing to do is to set $T_1 = \mathcal{H}(M^{(1)}, H^{(1)})$, where $\mathcal{H}$ is a collision-resistant hash function. Similarly, $T_i = \mathcal{H}(M^{(i)}, H^{(i)})$, because router P_i uses the same $\mathsf{ProcOnion}$ procedure as router P_1. Therefore, to compute C_1, the sender first needs to generate the keys $(K_1, \ldots, K_{n+1})$, then compute $(C_2, \ldots, C_{n+1})$. Then the sender will have enough information to obtain the tag T_1 and to compute C_1.

So, let us figure out how to compute O_2. Consider how P_1 will process O_1 (adapting Construction 1):

$$
\begin{aligned}
(O_2, P_2) &= \mathsf{ProcOnion}(SK(P_1), O_1, P_1) \\
&= (M^{(2)}, H^{(2)}, C_2, P_2) \\
&= (\{M^{(1)}\}_{K_1^{-1}}, (R_1, \{H_3^{(1)}\}_{K_1^{-1}}, \{H_2^{(1)}\}_{K_1^{-1}}), \{H_1^{(1)}\}_{K_1^{-1}}) \\
&= (\{\{\{m\}_{K_4}\}_{K_3}\}_{K_2}, (R_1, \{\{C_4, P_4\}_{K_3}\}_{K_2}, \{C_3, P_3\}_{K_2}\}), C_2, P_2)
\end{aligned}
$$

We need to address how the value R_1 is formed. On the one hand, we have already established (in Construction 1) that it needs to be random-looking, as we need to make sure that P_2 does not realize that R_1 is a padding, rather than a meaningful header. On the other hand, consider the ciphertext $C_2 \leftarrow E(PK(P_2), K_2, T_2)$, where, as we have established $T_2 = \mathcal{H}(M^{(2)}, H^{(2)})$. So, as part of the header $H^{(2)}$, the value R_1 needs to be known to the sender at $\mathsf{FormOnion}$ time, to ensure that the ciphertext C_2 is formed using the correct tag T_2. Thus, let us set R_1 *pseudorandomly*, as follows: $R_1 = \{P_1 \circ 0^{\ell_C}\}_{K_1^{-1}}$, where recall that ℓ_C is the number of bits required to represent the ciphertext C_1. Similarly, $R_i = \{P_i \circ 0^{\ell_C}\}_{K_i^{-1}}$. (Why include the value P_i into the pad? This is something we need to make the proof of security go through. Perhaps it is possible to get rid of it somehow.)

Now we can explain how $\mathsf{FormOnion}$ works (still using $N = 4$, $n = 3$): pick symmetric keys (K_1, K_2, K_3, K_4). Let $R_i = \{P_i \circ 0^{\ell_C}\}_{K_i^{-1}}$ for $1 \leq i \leq 4$. First, form the innermost onion O_4, as follows:

$$O_4 = (\{m\}_{K_4}, (R_3, \{R_2\}_{K_3^{-1}}, \{\{R_1\}_{K_3^{-1}}\}_{K_2^{-1}}), C_4 \leftarrow E(PK(P_4), K_4, T_4))$$

where recall that $T_4 = \mathcal{H}(M^{(4)}, H^{(4)})$. Now, for $1 < i \leq 4$, to obtain O_{i-1} from $O_i = (M^{(i)}, (H_3^{(i)}, H_2^{(i)}, H_1^{(i)}), C_i)$, let

$$\begin{aligned} M^{(i-1)} &= \{M^{(i)}\}_{K_{i-1}} & H_3^{(i-1)} &= \{H_2^{(i)}\}_{K_{i-1}} \\ H_2^{(i-1)} &= \{H_1^{(i)}\}_{K_{i-1}} & H_1^{(i-1)} &= \{C_i, P_i\}_{K_{i-1}} \\ T_{i-1} &= \mathcal{H}(M^{(i-1)}, H^{(i-1)}) & C_{i-1} &\leftarrow E(PK(P_{i-1}), K_{i-1}, T_{i-1}) \end{aligned}$$

It is easy to verify that the onions (O_1, O_2, O_3, O_4) formed this way will satisfy the correctness property (Definition 3).

We are now ready to describe our construction more formally. Note that without the intuition above, the more formal description of our construction may appear somewhat terse.

Setup. The key generation/setup algorithm G for a router is as follows: run $Gen(1^k)$ to obtain (PK, SK). Router name P must be a string of length $2\ell_K$, chosen uniformly at random by a trusted source of randomness; this needs to be done so that even for a PK chosen by an adversary, the name P of the corresponding router is still a random string. (In the random oracle model, this can be obtained by querying the random-oracle-like hash function on input PK.) Register (P, PK) with the PKI.

Forming an onion. On input message $m \in \{0,1\}^{\ell_m}$, a set of router names $(P_1, \ldots, P_{n+1})$, and a set of corresponding public keys $(PK_1, \ldots, PK_{n+1})$, the algorithm $\mathsf{FormOnion}$ does:

1. (Normalize the input). If $n + 1 < N$, let $P_i = P_{n+1}$, and let $PK_i = PK_{n+1}$ for all $n + 1 < i \leq N$.
2. (Form inner layer). To obtain the inner onion O_N, choose symmetric keys $K_i \leftarrow \{0,1\}^{\ell_K}$, for $1 \leq i \leq N$. Let $R_i = \{P_i \circ 0^{\ell_C}\}_{K_i^{-1}}$. Let $M^{(N)} = \{m\}_{K_N}$. As for the header, $H_{N-1}^{(N)} = R_{N-1}$, $H_{N-2}^{(N)} = \{R_{N-2}\}_{K_{N-1}^{-1}}$, and,

in general, $H_i^{(N)} = \{\ldots\{R_i\}_{K_{i+1}^{-1}}\ldots\}_{K_{N-1}^{-1}}$ for $1 \leq i < N-1$. Let $T_N = \mathcal{H}(M^{(N)}, H_{N-1}^{(N)}, \ldots, H_1^{(N)})$. Finally, let $C_N \leftarrow E(PK_N, K_N, T_N)$. Let $O_N = (M^{(N)}, H_{N-1}^{(N)}, \ldots, H_1^{(N)}, C_N)$.

3. (Adding a layer). Once $O_i = (M^{(i)}, H_{N-1}^{(i)}, \ldots, H_1^{(i)}, C_i)$ is computed for any $1 < i \leq N$, compute O_{i-1} as follows: $M^{(i-1)} = \{M^{(i)}\}_{K_{i-1}}$; $H_j^{(i-1)} = \{H_{j-1}^{(i)}\}_{K_{i-1}}$ for $1 < j \leq N$; $H_1^{(i-1)} = \{P_i, C_i\}_{K_{i-1}}$. Let $T_{i-1} = \mathcal{H}(M^{(i-1)}, H_{N-1}^{(i-1)}, \ldots, H_1^{(i-1)})$. Finally, let $C_{i-1} \leftarrow E(PK_{i-1}, K_{i-1}, T_{i-1})$. The resulting onion is $O_{i-1} = (M^{(i-1)}, H_{N-1}^{(i-1)}, \ldots, H_1^{(i-1)}, C_{i-1})$.

Processing an onion. On input a secret key SK, an onion $O = (M, H_N, \ldots, H_1, C)$, and the router name P, do: (1) compute tag $T = \mathcal{H}(M, H_N, \ldots, H_1)$; (2) let $K = D(SK, C, T)$; if $K = \perp$, reject; otherwise (3) let $(P', C') = \{H_1\}_{K^{-1}}$; (4) if P' does not correspond to a valid router name, output $(\{M\}_{K^{-1}}, \perp)$ (that means that P is the recipient of the message $m = \{M\}_{K^{-1}}$); otherwise (5) send to P' the onion $O' = (\{M\}_{K^{-1}}, \{P \circ 0^{\ell_C}\}_{K^{-1}}, \{H_N\}_{K^{-1}}, \ldots, \{H_2\}_{K^{-1}}, C')$

Theorem 2. *The construction described above is correct, achieves integrity, and is onion-secure in the PKI model where each router's name is chosen as a uniformly random string of length $2\ell_K$, and assuming that (1) (Gen, E, D) is a CCA-2 secure encryption with tags; (2) p is a PRP simultaneously secure for block lengths ℓ_M and ℓ_H for which the non-standard assumption holds, and (3) hash function $\mathcal{H}$ is collision-resistant.*

Proof. (Sketch) Correctness follows by inspection. Integrity is the consequence of our non-standard assumption: Suppose that our goal is to break the non-standard assumption. So we are given as input two strings P_1' and P_2'. We set up the the set of honest players $\mathcal{Q}$, together with their key pairs, as in Definition 2, giving each player a name chosen at random and assigning the strings P_1' and P_2' as names for two randomly chosen routers. Note that as our reduction was the one to set up all the keys for the honest routers, it is able to successfully answer all $\mathsf{ProcOnion}$ queries on their behalf, as required by Definition 4. Suppose the adversary is capable of producing an onion whose path is longer than N. With probability $1/|\mathcal{Q}|$, this onion O_1 is sent to router $P_1 = P_1'$. Let $\{(P_1, O_1, K_1), \ldots, (P_i, O_i, K_i), \ldots\}$ be the evolution of this onion augmented by the symmetric keys $(K_1, \ldots, K_i, \ldots)$ that router P_i obtains while running $\mathsf{ProcOnion}(SK(P_i), O_i, P_i)$. According to our $\mathsf{ProcOnion}$ construction, the value (if any) that router P_N obtains as a candidate for $(P_{N+1} \circ C_{N+1})$ is the string $\{H_1^{(N)}\}_{K_N^{-1}} = \{\{\ldots\{P_1 \circ 0^{\ell_C}\}_{K_1^{-1}}\ldots\}_{K_{N-1}^{-1}}\}_{K_N^{-1}} = P \circ C$. For this to be a valid onion O_{N+1}, P must be a valid router name. If $P = P_1$, then we have broken our assumption. Otherwise $P \neq P_1$, but then with probability at least $1/|\mathcal{Q}|$, $P = P_2'$ and so we also break the non-standard assumption.

It remains to show onion-security. First, we use a counting argument to show that, with probability $1 - 2^{-\ell_K + \Theta(\log|\mathcal{Q}|)}$ over the choice of router names, the adversary cannot re-wrap the challenge onion.

Suppose that the challenger produces the onion layers $(O_1, \ldots, O_j)$. Consider the header $H_{N-1}^{(j)}$ of the onion O_j. By construction, $H_{N-1}^{(j)} = R^{(j-1)} = \{P_{j-1} \circ 0^{\ell_C}\}_{K_{j-1}^{-1}}$. Also by construction, any SK, $O' = (M', H', C')$ and P' such that $O_j = \mathsf{ProcOnion}(SK, O', P')$ must satisfy $\{P' \circ 0^{\ell_C}\}_{(K')^{-1}} = H_{N-1}^{(j)}$, where K' is the decryption of C' under key SK. Thus, to re-wrap the onion, the adversary must choose P_{j-1}, P' and K' such that $\{P_{j-1} \circ 0^{\ell_C}\}_{K_{j-1}^{-1}} = \{P' \circ 0^{\ell_C}\}_{(K')^{-1}}$.

Let P be a router name, and let K be a key for the PRP p. Let

$$Bad(P, K) = \{P' \; : \; \exists K' \text{ such that } P' \neq P \wedge \{P \circ 0^{\ell_C}\}_{K^{-1}} = \{P' \circ 0^{\ell_C}\}_{(K')^{-1}}\} \; .$$

As there are at most 2^{ℓ_K} choices for K', and p is a permutation, for all (P, K), $|Bad(P, K)| \leq 2^{\ell_K}$. Let $Bad(\mathcal{Q}, K) = \{P' \; : \; \exists P \in \mathcal{Q} \text{ such that } P' \in Bad(P, K)\}$. Then $|Bad(\mathcal{Q}, K)| \leq |\mathcal{Q}| \max_P |Bad(P, K)| \leq |\mathcal{Q}| 2^{\ell_K}$.

Assume without loss of generality that the key K_{j-1} is fixed. Thus, for this onion to be "re-wrappable," it must be the case that there exists some $P' \in Bad(\mathcal{Q}, K_{j-1})$ that corresponds to a valid router name, i.e. $\mathcal{Q} \cap Bad(\mathcal{Q}, K_{j-1}) \neq \emptyset$. As any $P' \in \mathcal{Q}$ is chosen uniformly out of a set of size $2^{2\ell_K}$, while $|Bad(\mathcal{Q}, K_{j-1})| \leq 2^{\ell_K + \log |\mathcal{Q}|}$, it is easy to see that the probability over the choice of K_{j-1} and the router names for the set $\mathcal{Q}$ that the onion is "re-wrappable," is only $2^{-\ell_K + \Theta(\log |\mathcal{Q}|)}$.

It remains to show that no adversary can guess the challenger's bit b, provided (as we have shown) that it cannot re-wrap the onion. This proof follows the standard "sequence of games" [20] argument. Suppose that we set up the following experiments. In experiment (1), the challenger interacts with the adversary as in Definition 5 when $b = 0$, using $\mathsf{FormOnion}$. In experiment (2), the challenger departs from the first experiment in that it deviates from the usual $\mathsf{FormOnion}$ algorithm in forming the ciphertext C_j as $C_j \leftarrow E(PK, K', T_j)$, where $K' \neq K_j$ is an independently chosen key. It is easy to see that distinguishing experiments (1) and (2) is equivalent to breaking either the CCA2 security of the underlying cryptosystem, or the collision-resistance property of $\mathcal{H}$.

In experiment (3), the challenger forms O_j as follows: Choose keys $K_1, \ldots K_{j-1}$, and K'. Let $R_i = \{P_i \circ 0^{\ell_C}\}_{K_i^{-1}}$ for $1 \leq i < j$. $M^{(j)} \leftarrow \{0,1\}^{\ell_m}$, $H_i^{(j)} = \{\ldots \{R_i\}_{K_{i+1}^{-1}} \ldots\}_{K_{j-1}^{-1}}$ for $1 \leq i < j$, $H_i^{(j)} \leftarrow \{0,1\}^{\ell_H}$ for $j \leq i \leq N-1$. Finally, $C_j \leftarrow E(PK, K', T_j)$. The other onions, O_{j-1} through O_1, are formed using the "adding a layer" part of the $\mathsf{FormOnion}$ construction. It can be shown (omitted here for lack of space) that an adversary who can distinguish experiments (2) and (3) can distinguish p_{K_j} from a random permutation. The intuition here is that in experiment (3), everything that's supposed to be the output of p_{K_j} or $p_{K_j}^{-1}$ is random.

In experiment (4), the onion is formed by running $\mathsf{FormOnion}(r, (P_1, \ldots, P_j), (PK_1, \ldots, PK_j))$, *except* that C_j is formed as $C_j \leftarrow E(PK, K', T_j)$. Telling (3) and (4) apart is also equivalent to distinguishing p from a random permutation. The intuition here is that in experiment (4) the first $j-1$ parts of the header of onion O_j are formed as in experiment (3), while the rest are formed differently, and permuted using key K_j.

Finally, experiment (5) does what the challenger would do when $b = 1$. It is easy to see that distinguishing between (4) and (5) is equivalent to breaking CCA2 security of the cryptosystem or collision-resistant of $\mathcal{H}$.

4.2 Response Option

Suppose that P_s wants to send an anonymous message m to P_r and wants P_r to be able to respond. Our construction allows for that possibility (however we omit the definition and proof of security).

The sender chooses a path $(P'_1, \ldots, P'_n)$ for the return onion, (so $P'_0 = P_r$, and $P'_{n+1} = P_s$). Next, the sender forms $(O'_1, \ldots, O'_{n+1}) = \mathsf{FormOnion}(\varepsilon, (P'_1, \ldots, P'_{n+1}), (PK(P'_1), \ldots, PK(P'_{n+1})))$. It then chooses a symmetric authentication and encryption key a and remembers all the keys $(K'_1, \ldots, K'_{n+1})$ used during $\mathsf{FormOnion}$. Finally, it forms its message as $m' = m \circ a \circ O'_1 \circ P'_1$, and forms its actual onion in the usual way, i.e., chooses intermediate routers $(P_1, \ldots, P_n)$ and sets $(O_1, \ldots, O_{n+1}) \leftarrow \mathsf{FormOnion}(m', (P_1, \ldots, P_n, P_r), (PK(P_1), \ldots, PK(P_n), PK(P_r)))$.

Upon receipt of $m' = (m, a, O'_1, P'_1)$, P_s responds as follows. Suppose his response is M. He encrypts and authenticates M using a, forming a ciphertext c_1. He then sends (c_1, O'_1) to P'_1, with the annotation that this is a response onion. A router P receiving a message (c, O') with the annotation that this is a response onion, applies $\mathsf{ProcOnion}$ to onion O' only, ignoring c. Recall that as a result of this, P' obtains (O'', P'') (what to send to the next router and who the next router is) and the key K'. It then sends the values $(\{c\}_{K'}, O'')$ to P'', also with the annotation that this is a response onion. Eventually, if all goes well, the tuple $(\{\ldots\{c_1\}_{K'_1}\ldots\}K'_n, O'_{n+1})$ reaches P_s, who, upon processing O'_{n+1} recognizes that he is the recipient of this return onion, and is then able to obtain c_1 using the keys $K'_1, \ldots, K_n$ it stored, and to validate and decrypt c_1 using the key a. Note that, due to the symmetric authentication step using the key a, if P_r is honest, then no polynomial-time adversary can make P_s accept an invalid response.

5 Acknowledgments

We thank Ron Rivest for pointing out that our cryptographic definition must guarantee onion-integrity in addition to correctness and security. We are grateful to Leo Reyzin for valuable discussions. We thank the anonymous referees for their thoughtful comments. Jan Camenisch is supported by the IST NoE ECRYPT and by the IST Project PRIME, which receive research funding from the European Community's Sixth Framework Programme and the Swiss Federal Office for Education and Science. Anna Lysyanskaya is supported by NSF CAREER Grant CNS-0374661.

References

1. M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *TCC 2004*, vol. 2951 of *LNCS*, pp. 336–354.
2. O. Berthold, A. Pfitzmann, and R. Standtke. The disadvantages of free MIX routes and how to overcome them. In *Proceedings of Designing Privacy Enhancing Technologies*, vol. 2009 of *LNCS*, pp. 30–45. Springer-Verlag, July 2000.
3. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology — CRYPTO 2003*, LNCS, 2003.
4. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 136–145, 2001.
5. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT 2003*, vol. 2656 of *LNCS*, pp. 255–271. Springer Verlag, 2003.
6. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, Feb. 1981.
7. D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, Oct. 1985.
8. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.
9. G. Danezis. The traffic analysis of continuous-time mixes. In *Privacy Enhancing Technologies (PET)*, 2004.
10. R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pp. 303–320. USENIX, 2004.
11. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM Journal on Computing*, 2000.
12. D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion routing for anonymous and private internet connections. *Comm. of the ACM*, 42(2):84–88, Feb. 1999.
13. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, Apr. 1984.
14. D. Kesdogan, D. Agrawal, and S. Penz. Limits of anonymity in open environments. In *Information Hiding 2003*, vol. 2578 of *LNCS*, pp. 53–69. Springer, 2003.
15. M. Luby and C. Rackoff. How to construct pseudorandom permutations and pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, Apr. 1988.
16. B. Möller. Provably secure public-key encryption for length-preserving Chaumian mixes. In *Cryptographer's Track — RSA 2003*, pp. 244–262. Springer, 2003.
17. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Research in Security and Privacy*, pp. 184–200. IEEE Computer Society Press, 2001.
18. M. K. Reiter and A. D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998.
19. V. Shoup. A proposal for an ISO standard for public key encryption. http://eprint.iacr.org/2001/112, 2001.
20. V. Shoup. Sequences of games: a tool for taming complexity in security proofs. http://eprint.iacr.org/2004/332, 2004.
21. V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *EUROCRYPT '98*, vol. 1403 of *LNCS*. Springer, 1998.
22. D. Wikström. A universally composable mix-net. In *Theory of Cryptography Conference*, vol. 2951 of *LNCS*, pp. 317–335. Springer, 2004.
23. Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao. On flow correlation attacks and countermeasures in mix networks. In *PET*, 2004.