Breaking and (Partially) Fixing Provably Secure Onion Routing

Christiane Kuhn*‡, Martin Beck†, Thorsten Strufe*†
*⟨firstname.lastname⟩@kit.edu, KIT Karlsruhe †⟨firstname.lastname⟩@tu-dresden.de, TU Dresden

***Abstract*—After several years of research on onion routing, Camenisch and Lysyanskaya, in an attempt at rigorous analysis, defined an ideal functionality in the universal composability model, together with properties that protocols have to meet to achieve provable security. A whole family of systems based their security proofs on this work. However, analyzing HORNET and Sphinx, two instances from this family, we show that this proof strategy is broken. We discover a previously unknown vulnerability that breaks anonymity completely, and explain a known one. Both should not exist if privacy is proven correctly.**

In this work, we analyze and fix the proof strategy used for this family of systems. After proving the efficacy of the ideal functionality, we show how the original properties are flawed and suggest improved, effective properties in their place. Finally, we discover another common mistake in the proofs. We demonstrate how to avoid it by showing our improved properties for one protocol, thus partially fixing the family of provably secure onion routing protocols.

I. Introduction

Anonymous communication protocols are developed to protect communication meta data from surveillance. With millions of users[1] Tor [16] is the most widely known protocol to restrict the information an adversary learns. It relies on the idea of Onion Routing (OR) [21]. This generic approach removes the relationship between a message and its corresponding sender by forwarding the message over multiple proxies that modify it at each hop.

With increasing importance of OR, the need to build efficient protocols for low delay communication with proven security guarantees became apparent. To simplify building those protocols, Sphinx [13] was proposed as a secure packet format for onions. Building on this format HORNET [11], among others, was proposed for high-speed OR at the network layer. Using multiple cryptographic techniques the authors both of Sphinx and HORNET present proofs along the lines of the strategy proposed by Camenisch and Lysyanskaya in [8].

This proof strategy is based on defining an *ideal functionality* for OR[2] in the universal composability (UC) model. The functionality is an abstraction to show which information even a perfect OR scheme leaks to an adversary. The authors in addition design protocol *properties*. Proving that a real world protocol complies with these properties, they claim, implies the security and privacy of their ideal OR functionality. This convenient proof scheme has been used to analyze the privacy of a whole family of recent, efficient packet formats (e.g. the improved Minx [29] and Sphinx [13]) and OR protocols (e.g. HORNET [11] and TARANET [12]).

Analyzing HORNET, we discovered a simple attack on its data transmission phase that allows it to link senders to receivers and large parts of the messages to their senders as well. Our attack complies to HORNET's adversary model and should have been detected when proving its security. We found that similar attacks are to some extent possible on related work [12], [13], [29]. In addition, there is a padding flaw in Sphinx [13], luckily detected and corrected in the implementation[3], that has escaped the formal analysis. Undetected, this flaw would have jeopardized the privacy of the senders in systems using Sphinx.

As all the protocols prove privacy based on the ideal functionality and properties of [8], attacks threatening the users' privacy should not be possible in the final protocol. We thus set out to identify and correct the mistakes in the process. As it turns out, there are multiple open questions and discrepancies that have to be solved for the proof strategy.

First, no one ever analyzed the privacy this ideal OR functionality actually achieves, to start with. Many papers [1], [3], [4], [6], [15], [17]–[19], [26]–[28], [30] citing it disagree violently on this point. As our first contribution towards solving the matter, we analyze the ideal functionality. We show that it indeed implies the privacy goals expected for OR, namely sender anonymity and relationship anonymity, against a limited yet useful adversary model.

Next, we look closer at the attack on Sphinx and realize the first mistake: The properties proposed to imply the privacy of the ideal functionality are not sufficient. Proving the original properties thus does not provide any privacy guarantee. Having a closer look at the properties, we discover that one of them is inexact, a second missing important aspects, and the last two lack to provide any additional privacy. To understand what exactly is missing, we construct two obviously broken protocols that still fulfill the properties. Based on our insights from the broken protocols, we construct two new properties, Tail-Indistinguishability and Layer-Unlinkability, and prove that they, together with the correction of the inexact property, indeed imply the privacy of the analyzed ideal functionality.

‡ This work in parts was carried out while affiliated with TU Dresden.

[1] according to https://metrics.torproject.org/userstats-relay-country.html

[2] Understanding of *OR* varied in the field. To be compliant with the terms of [8], we understand OR in this work as a free-route Chaumian MixNet [10] without requiring that messages are delayed. This conforms with the understanding of [21] and [16] except that circuits are excluded.

[3] https://github.com/UCL-InfoSec/sphinx/blob/c05b7034eaffd8f98454e0619b0b1548a9fa0f42/SphinxClient.py#L67

DOI 10.1109/SP40000.2020.00039

Thus, they allow to prove privacy with the convenient strategy of showing that a protocol meets the improved properties.

By reconsidering our new attack on HORNET, we uncover an independent second mistake: The properties of Camenisch and Lysyanskaya have not been proven correctly for the protocols. More precisely, the oracles used in the original definition of the properties have been ignored or weakened.

Finally, we demonstrate how to perform an analysis for our new properties, by proving that a variation of Sphinx [5], which improves performance but neglects replies, has (with the small additional correction to the padding flaw known from the Sphinx implementation) the privacy of the ideal functionality.

By solving the issues, it turns out that the model behind the ideal functionality does neither support anonymous reply packets, nor sessions – which are frequently adapted in above papers. The privacy for these extensions cannot be proven using the given ideal functionality. In this work, we favor a rigorous treatment of the foundations, i.e. sending a message to a receiver, over extensions like sessions and reply channels. We conjecture that with the solid foundation given in this paper the issues of sessions and replies can be solved in future work by adapting the functionality and properties.

Our main contributions are: (a) a privacy analysis of the ideal functionality of Camenisch and Lysyanskaya; (b) a rigorous analysis of the original properties; (c) the design of improved properties that provably achieve the ideal functionality; (d) a new attack on HORNET, that similarly is possible on the improved Minx (and in slight violation of their models, on TARANET and Sphinx); (e) demonstrations of flaws in the privacy proofs of the above named formats and systems; (f) a demonstration how to prove the corrected properties.

Outline: We first introduce the background, then analyze the ideal functionality, followed by the explanation of the Sphinx flaw and original properties. After this we show weaknesses of the original properties, construct new properties and prove them secure. Next, we explain the new HORNET attack and the flaw in the proofs. Finally, we prove a variation of Sphinx private, discuss our findings and conclude the paper.

II. Background

This section explains the adversary model, OR and selected existing systems based on OR. We further introduce the formal proof strategy [8] and the used privacy definitions [23].

For explaining OR, we consider the scenario of whistleblower Alice who wants to leak sensitive information to media agent Bob and uses open anonymization systems (like Tor) to hide from a regime that deploys mass surveillance.

A. Adversary Model

Assuming a nation state adversary we have to expect a global attacker with full control over the Internet infrastructure. This entails the possibility to observe all links and to actively drop, delay, modify, and insert packets on any link. Given the open nature of anonymization systems, the adversary can easily provide a large subset of nodes, which seemingly run the anonymization system, but are really under her full control. She hence knows all secret keys of those nodes, and she can modify, drop, and insert packets at each of them. Even the receivers are untrusted and considered potentially under control of the adversary, and as the system is open, the adversary may also act as one or a set of senders, seemingly using the anonymization system parallel to Alice. We assume full collusion between all adversarial parties, but follow the common assumption that the attacker is limited to probabilistic polynomial time algorithms (PPT). These assumptions are common for onion routing, and they correspond to the model in [8].

B. Onion Routing (OR)

Considering the scenario, sending her message to Bob, the journalist, Alice requires that both Bob and the regime shall not learn that she was the individual sending the message. Given the existence of a trusted proxy, she can encrypt her message with the public key of the proxy and send it there, to be decrypted and forwarded to Bob on her behalf. Her identity then is hidden in the set of all users that communicate over this proxy at the same time. The set of these users is commonly called her *anonymity set*.

Given the open nature of the system, Alice cannot trust any single proxy completely. She hence chooses a chain of proxies, hoping that one of the proxies is honest and does not collaborate with the adversary. To hide the mapping between the packets that arrive at and depart from a proxy, she consecutively encrypts the packet for each of the proxies on the chain, and includes a header signaling where to forward the packet next. Each proxy locally decrypts and forwards the packet. The last proxy decrypts it to the original message and forwards it to Bob.

As the packet is encrypted in several layers that consecutively are removed, the scheme is commonly called *onion encryption*. The proxies hence often are called *onion routers*, or *relays*.

Decrypting at the relays yields the intermediate header and a shorter onion for the next relay. Corresponding length reductions of the onions would leak information that the adversary could use to link observed packets arriving and departing at an honest relay. Onions hence are usually padded to a fixed length that is globally known, which restricts the maximum length of the payload as well as the number of relays on the path that can be addressed. We therefore assume the maximum path length N in terms of hops between an honest sender and a receiver.

Assumption 1: The OR protocol has a maximum path length of N.

Protection in OR follows from hiding the link between incoming and outgoing onions at a relay. Should the adversary by chance control all proxies that are chosen for an onion, she can trivially reversely link outgoing to incoming onions for all relays, and hence identify Alice as the original sender of a message delivered to Bob. As the path is chosen by Alice who actively aims to remain anonymous towards Bob and the regime, she will pick a path solely containing corrupted relays only rarely, by mistake. We therefore, deem it suitable to add the following assumption for our analysis:

Assumption 2: There is at least one honest relay on the chosen path, if the sender is honest.

Further, as the adversary can actively insert packets, she can replay the same onion at the honest relay and observe the same behavior twice. OR protocols hence usually implement a replay protection, by detecting and dropping replayed onions. For an easier analysis, we limit our scope to replay protection mechanisms that drop onions that have already been processed:

Assumption 3: The replay protection, if one is used, drops bit-identical onions.

C. Network Model

Onion Routing can be used in two different modes: the receiver participating in the anonymization protocol, or not. The first case considers an *integrated system* to be set up for anonymous communication. The receiver will act as an onion router and, while processing an onion, discover that it is intended for herself. In the second case messages are anonymized as a *service* and the receiver is unaware of the anonymization happening. The last router, called *exit node*, discovers that the message needs to be forwarded outside the anonymization network to reach its receiver.

D. Existing Schemes and Systems

Danezis and Goldberg [13] define *Sphinx*, a packet format for secure OR. Sphinx's goals are to provide bitwise unlinkability between onion layers before and after an honest node, resistance against all active tagging attacks to learn the destination or content of a message, and space efficiency. Hiding the number of hops an onion already traveled, and the indistinguishability of both forward onions as well as response onions on a reply channel were considered to further strengthen privacy. Their network model assumes anonymization services, and their adversary model mainly matches the above description. Traffic analysis, flooding or denial of service are however excluded. Tagging attacks, i.e. an adversary modifying onions before reinjecting them, on the other hand are explicitly allowed.

Sphinx's onion layers consist of a header that contains all path information except the receiver, and a payload that contains the protected message and protected receiver address. Padding and multiple cryptographic primitives are used for construction and processing of onions, but the integrity of the payload at each layer is not protected by Sphinx as this would conflict with their support for replies. Tampering with the payload is only recognized at the exit node. As security proof, Danezis and Goldberg prove the properties of [8] for Sphinx.

Predecessors to Sphinx were *Minx* [14] and its fixed version [29]. Like Sphinx, neither of the two protects the integrity of the payload at the relays. Beato et al. proposed a *variant of Sphinx* [5] that neglects replies and replaces the cryptographic primitives to increase performance and security, and thereby protects the integrity of the payload at each relay.

Subsequent to the work on packet formats, Chen et al. proposed the protocol *HORNET* [11] as a high-speed, highly scalable anonymity system for the network layer. The authors claim that HORNET protects the anonymity of Alice against a slightly restricted adversary compared to our attacker: The attacker does actively control a fraction of the relays (including the receiver), but corruption of links is not explicitly mentioned. Further, traffic analysis attacks are excluded as in the case of Sphinx. They assume an integrated anonymization network including the receiver. HORNET distinguishes between a setup phase and a transmission phase. It adapts Sphinx for the setup phase to create an anonymous header that allows for routing data in the subsequent transmission phase. Multiple cryptographic primitives are used in the construction and processing of packets in both phases. Similar to Sphinx, HORNET's data transmission phase does not protect the integrity of the payload at each relay. Further, at every relay the payload is decrypted with a block cipher in CBC mode.

Extending HORNET to protect against partially stronger adversaries, *TARANET* [12] bases its setup on Sphinx as well. Additionally, it proposes packet-splitting as a traffic-shaping technique to withstand some traffic-analysis. Therefore, however, shared trust between sender and receiver is presumed.

The privacy of HORNET's and TARANET's setup phase is claimed to follow from Sphinx. The privacy of their data transmission phase is proven following the same proof technique from [8], similar as in the improved Minx [29] and Sphinx.

E. Formally treating OR

Towards rigorous analysis of OR, Camenisch and Lysyanskaya [8] specified an ideal functionality in the UC framework and defined properties to ease the analysis of OR protocols[4].

1) UC Framework [9]: An *ideal functionality* in the UC framework is an abstraction of a real protocol that expresses the security and privacy properties as required in the real protocol. Proving that the real protocol realizes the ideal functionality implies proving that attacks on the real protocol do not reveal anything to the adversary she would not learn from attacks on the ideal functionality.

2) Formal OR Scheme: To model OR, [8] defines an *Onion Routing Scheme* as the set of three algorithms:

- Key generation algorithm G: $(PK, SK) \leftarrow G(1^\lambda, p, P)$ with public key PK, secret key SK, security parameter λ, public parameter p and router name P
- Sending algorithm $\mathrm{FormOnion}$: $(O_1, ..., O_{n+1}) \leftarrow \mathrm{FormOnion}(m, (P_1, \ldots, P_{n+1}), (PK_1, \ldots, PK_{n+1}))$ with O_i being the onion layer to process by router P_i, m the message, and PK_i the public key belonging to router P_i
- Forwarding algorithm $\mathrm{ProcOnion}$: $(O', P') \leftarrow \mathrm{ProcOnion}(SK, O, P)$ with O' the processed onion that is forwarded to P' and P the router processing O with secret key SK. O' and P' attains $\perp$ in case of error or if P is the recipient.

[4]Although designed for the integrated system model, it applies to the service model as well (except for renaming *recipient* to *exit node*) if no protection outside of the OR protocol exists. There the ideal functionality however only considers the anonymization network and additional private information might leak when the packet is sent from the exit node to the receiver.

3) Properties: [8] defines three *security properties* for OR schemes and proves that those imply realizing their ideal OR functionality, i.e. being private and secure. Later works [11]–[13] split one of the properties in two. The resulting four properties are Onion-Correctness, Onion-Integrity, Onion-Security and Wrap-Resistance:

Onion-Correctness requires that all messages use the intended path and reach the intended receiver in absence of an adversary. *Onion-Integrity* limits the number of honest relays that any onion (even one created by the adversary) can traverse. *Onion-Security* states that an adversary observing an onion departing from an honest sender and being routed to an honest relay, cannot distinguish whether this onion contains adversarial chosen inputs or a random message for the honest relay. The adversary is even allowed to observe the processing of other onions at the honest relay via an oracle. *Wrap-Resistance* informally means that an adversary cannot create an onion that after processing at a relay equals an onion she previously observed as an output at another relay, even if she has full control over the inputs.

F. Analysis Framework

We use the framework of Kuhn et al. [23] that unifies the privacy goals of existing theoretical analysis frameworks like AnoA [2] and others [7], [20], [22]. It introduces a well-analyzed hierarchy of privacy goals and thus allows our analysis results for OR to be easily comparable.

1) Privacy Goals: The idea of the analysis follows game-based security-proofs. It challenges an adversary to distinguish two simulations of the protocol that differ only in protected parts of the communications (e.g. who the sender of a certain message was). Each communication in this context contains a sender, receiver, message and auxiliary information, like, for our purpose, the path included in the onion. The communications input for the two simulations are called scenarios. They are freely chosen by the adversary to reflect the worst case. *Privacy notions* specify formally in which elements the scenarios are allowed to differ, or, in other words, which information has to be protected by the protocol.

Four privacy notions are of specific interest when analyzing OR. The first is a strong form of confidentiality: The adversary is unable to decide which of two self-chosen messages was sent in the simulation. As thus the message is unobservable, this notion is called *Message Unobservability* ($M\overline{O}$).

The second corresponds to our example above, and is a form of sender anonymity: Informally speaking, the adversary is unable to decide, which of the messages that she provided is sent by which of the senders that she chose. As thus she cannot link the sender to its message, this notion is called *Sender-Message Unlinkability* ($SM\overline{L}$).

The third, conversely, is a form of receiver anonymity: The adversary is unable to decide, which of the messages that she provided is received by which of the receivers that she chose. As thus she cannot link the receiver to its message, this notion is called *Receiver-Message Unlinkability* ($RM\overline{L}$).

The fourth is a form of relationship anonymity: The adversary is unable to decide which pairs of two self-chosen senders and two self-chosen receivers communicate with each other. As thus she cannot link the sender to the receiver, this notion is called *Sender-Receiver Unlinkability* ($SR\overline{L}$).[5]

2) Adversary: All the privacy notions can be analyzed for different user (sender and receiver) corruption. Therefore, options for user corruption are defined and added to the abbreviation of privacy notion X:

X_0: no users are corrupted, but some relays or links can be,
X_s: only receivers, relays, and links can be corrupted, but no senders,
X_e: senders, receivers, relays, and links can be corrupted (some limitations apply to prevent the adversary to trivially win the game)

The framework introduces adversary classes as part of the game, known to the adversary. They specify modifications of the input from, as well as the output to the adversary. Their purpose is to fine-tune the adversary capabilities e.g. to make sure that Assumption 2 is met in the scenarios the adversary is challenged to distinguish.

3) Relation of Goals: Analyzing OR we are interested in determining the strongest notion that it achieves. The analysis in the framework then allows statements even for notions that are not directly analyzed, as it proves a hierarchy: By showing that a certain notion is achieved, all implied (weaker) notions are shown to be achieved as well.

Given the claims in [8], [11], [13], we are specifically interested in the above mentioned notions of sender- as well as receiver-message unlinkability ($SM\overline{L}$ and $RM\overline{L}$), which each implies sender-receiver unlinkability ($SR\overline{L}$), and the independent message unobservability ($M\overline{O}$) (See Appendix A for detailed definitions. For the analyses of other notions we refer the interested reader to the extended version of this paper at [24]).

III. Analyzing the Ideal OR Functionality

There indeed is confusion about which privacy the ideal functionality $\mathcal{F}$ of [8] actually guarantees. The work itself states only that "its not hard to see that $\mathcal{Z}$ [the environment, a construct of the UC Framework that gets all observations of the adversary] 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."

[1], [3], [27], [28], [30] state that this translates to the degree of anonymity Tor provides, although [15], [18] argue that it is not applicable for Tor. [4] states that it "hide(s) the source and destination over a network," [26] even understood it as "a concrete ZK proof of senders' knowledge of their messages" and [6] as "provable reduction from unlinkability to traffic analysis." [19] states that the privacy is "that an adversary cannot correctly guess relations between incoming messages

[5]This notion is called $(SR)\overline{L}$ in [23].

and outgoing messages at onion routers, and [...] that each onion router cannot know the whole route path of any onion." While [18] and [17] realize that the anonymity is not analyzed and suspect it to be close to the one of [25], which claims to have sender and receiver anonymity against a global passive adversary [17].

We hence set out to analyze the actual privacy guarantees of the ideal functionality.

A. Ideal Functionality $\mathcal{F}$

Recall the basic idea of OR: an adversary can only track the communication from the sender until the first honest relay. After this she can no longer link the onion to the sender (or the route before the honest relay). Further, any onion layer does hide the included message and remaining path, as they are encrypted.

The ideal functionality for OR of [8] therefore uses temporary random IDs in place of onion packets. All network information necessary to create onions (sender, receiver, path, message, hopcount, a randomly chosen session ID) are stored within the ideal functionality, inaccessible to the adversary.

Sending the onion along a path of relays is represented by informing all relays about the temporary IDs of the corresponding onions they receive. The temporary ID is replaced with a new randomly drawn ID at every honest node.

The adversary in this model learns the temporary IDs on links and at the corrupted relays, and if the receiver is corrupted also the corresponding plaintext message. She specifically does not learn which departing ID at an honest relay corresponds to which received ID. The adversary however is allowed to decide when an ID is delivered to the next relay (and thus whether it is delivered at all), as she is assumed to control all links.

Nitpicking, we add a small detail to the ideal functionality as suggested by Camenisch and Lysyanskaya: The functionality represents the case of an honest sender well. However, for a corrupted sender the adversary trivially learns the complete path and message as the sender chooses it. As no secure protocol can remove information an adversary already knows, we add that the functionality outputs all information about the onion (sender, receiver, path, etc.) together with the temporary ID, if its sender is corrupted. The ideal functionality is detailed in Algorithm 1.

B. Analysis under Restricted Adversary Model

The ideal functionality was designed to capture the cryptographic properties of onion routing. Therefore, it does not protect against dropping or delaying onions. Hence, for this analysis we need to exclude attacks that result in dropping or delaying onions.[6] Given this adversary model[7] we are able to prove the privacy goals expected for OR.

1) Instantiation of the Framework: As the path $\mathcal{P}$ is an important input to an onion, we model it specified in the auxiliary information of a communication. The communications, including the auxiliary information, are picked arbitrarily by the adversary in the framework. Assumption 2 however requires at least one honest relay to exist on the path for our analysis. For this reason, we define the adversary class $\mathcal{C}$ to modify the path: $\mathcal{C}$ replaces the paths as chosen by the adversary with alternative paths, whenever an honest sender constructing the onion. The replacements are chosen at random from the set of paths with valid length that include at least one common honest relay.

We further restrict the adversary to be incapable of timing-based traffic analysis. Hence, in the traffic analysis restricted adversary class $\mathcal{C}$ the adversary must not use any timing information about the onion, i.e. the adversary class shuffles all the outputs from the ideal functionality for communications that are processed together before handing them to the adversary. Since the adversary is incapable of traffic analysis, the adversary class prohibits to delay packets. To further prohibit replay attacks, which we consider as special kind of traffic analysis attack, the adversary class drops any duplicated deliver requests from the adversary.

2) Analysis: Recall, the ideal functionality only outputs the message to the adversary for a corrupted receiver or sender. So, the message is protected if sender and receiver are honest or corrupted users get the same messages in both scenarios (limitation in X_e) and confidentiality $M\overline{O}$ is achieved.

Due to the adversary class $\mathcal{C}$, the adversary observes all outputs corresponding to the inputs of an honest relay in random order. Combined with random ID replacement, this prevents the adversary from linking departing onions to their received counterparts. However, it can still be observed that a user is actively sending if she has not previously received an onion (or: that a user is receiving, if upon receiving an onion she subsequently does not send one). This leads to Theorem 1, which we prove in our extended version [24].

Theorem 1: $\mathcal{F}$ achieves $M\overline{O}_e$, $SM\overline{L}_s$ and $RM\overline{L}_0$, and those implied by them, but no other notions of [23] for $\mathcal{C}$.

Note that under this adversary model sender anonymity ($SM\overline{L}$) is achieved even if the receiver is corrupted. From the hierarchy of [23], we know that this strong version of sender anonymity also implies relationship anonymity ($SR\overline{L}$). Note further that the receiver anonymity ($RM\overline{L}$) is only achieved if neither the sender nor the receiver is compromised. Thus, as soon as the sender is corrupted, receiver anonymity is no longer achieved.

C. First Summary

We have seen that the ideal functionality indeed provides the privacy expected from OR. Showing that a system realizes the ideal functionality proves these privacy notions for an adversary that cannot do timing-based traffic analysis. Even if in practice stronger adversary models are assumed, proving the realization of the ideal functionality is a useful way to reduce the problem

[6]However, we include modification attacks that do not lead to dropping or delaying onions, like classical tagging attacks. A protocol realizing the ideal functionality might either drop modified onions or keep them in the network, but prevent the attacker from learning critical information from them (i.e. the modified onion's path and message have no correlation to the original one's).

[7]This limitation is not significant in practical implementations as they need to employ additional protection against privacy attacks based on dropping and delaying onions.

Algorithm 1: Ideal Functionality $\mathcal{F}$

Data structure:
Bad: Set of Corrupted Nodes
L: List of Onions processed by adversarial nodes
B_i: List of Onions held by node P_i
// Notation:
// $\mathcal{S}$: Adversary (resp. Simulator)
// $\mathcal{Z}$: Environment
// $\mathcal{P} = (P_{o_1}, \ldots, P_{o_n})$: Onion path
// $O = (sid, P_s, P_r, m, n, \mathcal{P}, i)$: Onion = (session ID, sender, receiver, message, path length, path, traveled distance)
// N: Maximal onion path length

On message Process_New_Onion$(P_r, m, n, \mathcal{P})$ *from* P_s
- // P_s creates and sends a new onion (either instructed by $\mathcal{Z}$ if honest or $\mathcal{S}$ if corrupted)
- **if** $|\mathcal{P}| > N$; // selected path too long **then**
  - Reject;
- **else**
  - $sid \leftarrow^R$ session ID ; // pick random session ID
  - $O \leftarrow (sid, P_s, P_r, m, n, \mathcal{P}, 0)$; // create new onion
  - Output_Corrupt_Sender$(P_s, sid, P_r, m, n, \mathcal{P}, \text{start})$;
  - Process_Next_Step(O);

Procedure Output_Corrupt_Sender$(P_s, sid, P_r, m, n, \mathcal{P}, temp)$
- // Give all information about onion to adversary if sender is corrupt
- **if** $P_s \in$ *Bad* **then**
  - Send "$temp$ belongs to onion from P_s with $sid, P_r, m, n, \mathcal{P}$" to $\mathcal{S}$;

Procedure Process_Next_Step$(O = (sid, P_s, P_r, m, n, \mathcal{P}, i))$
- // Router P_{o_i} just processed O that is now passed to router $P_{o_{i+1}}$
- **if** $P_{o_j} \in$ *Bad for all* $j > i$; // All remaining nodes including receiver are corrupt **then**
  - Send "Onion from P_{o_i} with message m for P_r routed through $(P_{o_{i+1}}, \ldots, P_{o_n})$" to $\mathcal{S}$;
  - Output_Corrupt_Sender$(P_s, sid, P_r, m, n, \mathcal{P}, \text{end})$;
- **else**
  - // there exists an honest successor P_{o_j}
  - $P_{o_j} \leftarrow P_{o_k}$ with smallest k such that $P_{o_k} \notin$ Bad
  - $temp \leftarrow^R$ temporary ID;
  - Send "Onion $temp$ from P_{o_i} routed through $(P_{o_{i+1}}, \ldots, P_{o_{j-1}})$ to P_{o_j}" to $\mathcal{S}$;
  - Output_Corrupt_Sender$(P_s, sid, P_r, m, n, \mathcal{P}, temp)$;
  - Add $(temp, O, j)$ to L; // see Deliver_Message$(temp)$ to continue this routing

On message Deliver_Message$(temp)$ *from* $\mathcal{S}$
- // Adversary $\mathcal{S}$ (controlling all links) delivers onion belonging to $temp$ to next node
- **if** $(temp, \_, \_) \in L$ **then**
  - Retrieve $(temp, O = (sid, P_s, P_r, m, n, \mathcal{P}, i), j)$ from L;
  - $O \leftarrow (sid, P_s, P_r, m, n, \mathcal{P}, j)$; // jth router reached
  - **if** $j < n + 1$ **then**
    - $temp' \leftarrow^R$ temporary ID;
    - Send "$temp'$ received" to P_{o_j};
    - Store $(temp', O)$ in B_{o_j}; // See Forward_Onion$(temp')$ to continue
  - **else**
    - **if** $m \neq \perp$ **then**
      - Send "Message m received" to P_r

On message Forward_Onion$(temp')$ *from* P_i
- // P_i is done processing onion with $temp'$ (either decided by $\mathcal{Z}$ if honest or $\mathcal{S}$ if corrupted)
- **if** $(temp', \_) \in B_i$ **then**
  - Retrieve $(temp', O)$ from B_i;
  - Remove $(temp', O)$ from B_i;
  - Process_Next_Step(O);

of proving privacy to the attacks excluded by our adversary class $\mathcal{C}$.

IV. First Pitfall: Incomplete Properties

We first explain a known attack on Sphinx that should not be possible if Sphinx realizes the ideal functionality. Then we analyze the properties to see why the insecurity was not detected in the proof: the properties are incomplete and some of them do not increase privacy. We further generalize the attack on Sphinx and introduce an insecure protocol to make the shortcoming obvious and to help us in the construction of a new improved property. After that, we present a second independent insecurity, a corresponding broken protocol and again construct a new property to protect against it. Finally, we ensure that no more properties are missing by proving that they indeed imply the ideal functionality.

A. Attack on Sphinx

In Sphinx as presented in [13] the exit node receives β as part of the header. β contains the receiver address, an identifier, and a 0-bit string to pad β for the exit node to a fixed length. It is again padded with a filler string of random bits that compensates for the parts used to encrypt the earlier relay addresses. Further, the three components are XORed with the output of a pseudo-random number generator (PRNG).

The exit node hence can learn the length of the chosen path[8] with the following attack: The adversarial exit node observes (after XORing) where the first 1 bit after the identifier is. It knows that the filler string starts there or earlier and can determine by the length of the filler string a lower bound on the length of the path used.

Being able to know the length of the path is critical. If e.g. the routing topology is restricted or large parts of the path are only adversarial relays, this information limits the number of users under which the sender can hide and thus reduces her protection. According to the ideal functionality such an attack should not be possible if Sphinx, as proven with the properties of Camenisch and Lysyanskaya, realizes the ideal functionality.

B. Analyzing the Original Properties

In this section we have a closer look at the properties to see why the attack on Sphinx is not detected and we make four observations. The original definition of Onion-Correctness technically was not entirely correct, which we fix briefly. Integrity and Wrap-Resistance do not add privacy to the proposed combination of properties, at all. Onion-Security is required, but fails to protect against some weaknesses.

1) Onion-Correctness: Informally, *Onion-Correctness* requires that all messages use the intended path and reach the intended receiver in absence of an adversary:

Definition 1 (Original Onion-Correctness): Let $(G, \text{FormOnion}, \text{ProcOnion})$ be an OR scheme with

[8]To the best of our knowledge this flaw is only mentioned and corrected in the Sphinx implementation so far: https://github.com/UCL-InfoSec/sphinx/blob/c05b7034eaffd8f98454e0619b0b1548a9fa0f42/SphinxClient.py#L67

maximal path length N. Then for all polynomial numbers of routers P_i, for all settings of the public parameters p, for all $(PK(P), SK(P))$ generated by $G(1^\lambda, p, P)$, for all $n < N$, for all messages $m \in \mathcal{M}$, and for all onions O_1 formed as

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

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: $(m, \perp) = \text{ProcOnion}(SK(P_{n+1}), O_{n+1}, P_{n+1})$,

where $\mathcal{P}(O, P)$ returns the path included in O and $\mathcal{L}(O, P)$ the onion layers.

This however cannot be achieved by Sphinx or almost any other system suggested or implemented so far. They commonly use duplicate checks, which, practically implemented, may fail in a small number of cases (for example due to hash collisions) in reality. We hence allow the requirements 1) - 3) of the definition to fail with negligible probability, so that real systems can achieve Onion-Correctness at all.

2) Wrap-Resistance and Onion-Integrity: We analyzed Wrap-Resistance and Onion-Integrity and proved that they do not contribute anything to the privacy of a protocol that achieves Onion-Security and -Correctness.

We refer the interested reader to the extended version of this paper [24] for details. In short, we provide a template to add Wrap-Resistance and Onion-Integrity to any OR protocol that meets Onion-Security and Correctness, and prove that the template does not reduce what adversaries can learn.

3) Onion-Security: Onion-Security states that an adversary on the path between an honest sender and the next honest node (relay or receiver) cannot distinguish an onion that was created with her inputs (except for the keys of the honest node) from another one that contains a different message and is destined for this next honest node.

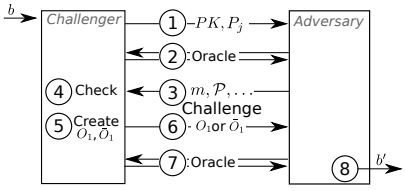

Fig. 1. Onion-Security game illustrated: Circled numbers represent the steps, and the boxes the parties of the game.

a) Game Structure: We illustrate the Onion-Security game in Fig. 1 and explain the steps informally first:

Basic Game (Step 1, 3 - 6, 8): Apart from an honest sender, Onion-Security assumes the existence of only a single honest relay (P_j). First in Step 1, the challenger chooses the name and public key of the honest node and sends it to the adversary. In the challenge starting in Step 3, the adversary is allowed to pick any combination of message and path as input choice of the honest sender, to model the worst case. In Step 4-6 the challenger checks that the choice is valid and if so, creates two onions $O_1, \bar{O}_1$ and sends one of them to the adversary depending on the challenge bit b. Finally in Step 8, the adversary makes a guess b' on which onion she received.

Adaptive and Modification Attacks (Step 2 and 7): So far the adversary only focused on one onion. However, a real adversary can act adaptively and observe and send modified onions to the honest node that she wants to bypass before and after the actual attack. Therefore, Onion-Security includes two oracle steps. To decide on her input and guess, the adversary is allowed to insert onions (other than the challenge onion) to the honest relay and observe the processed output as an oracle (Steps 2 and 7).

How the two onions $O_1, \bar{O}_1$ differ is illustrated in Fig. 2. O_1 is the first layer of the onion formed with the adversary chosen inputs, where the honest relay is at position j. In contrast, $\bar{O}_1$ is the first layer of the onion formed with the same path as O_1 except that the path ends at P_j as the receiver and a random message. The adversary can calculate the onion layers up to the honest relay based on the first layer. Onion-Security is achieved if the adversary is unable to distinguish whether the observed onion contains her chosen inputs or random content destined for the honest relay.

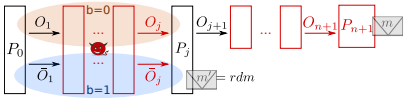

Fig. 2. Cases of Onion-Security illustrated: Red boxes represented corrupted relays, black boxes honest. The upper row of arrows represents the path of the onion O with inputs chosen by the adversary (message m received by P_{n+1}); the lower an onion $\bar{O}$ containing a randomly chosen message m' that takes the path to the honest relay P_j, only. For $b = 0$ the onion layers in the orange ellipse are observed by the adversary, i.e. the layers processed at $P_1..P_{j-1}$ of onion O. For $b = 1$ the layers in the blue ellipse are observed, i.e. the corresponding layers of $\bar{O}$. Notice that the adversary does not observe any output of P_j in this game.

b) Definition: Formally, the Onion-Security game is defined as follows:

Definition 2 (Original Onion-Security): Consider an adversary interacting with an OR challenger as follows.

1) The adversary receives as input a challenge public key PK, chosen by the challenger who generates $(PK, SK) \leftarrow G(1^\lambda, p, P_j)$, and the router name P_j.
2) The adversary submits any number of onions O_i of her choice to the challenger (oracle queries), and obtains the output of $\text{ProcOnion}(SK, O_i, P_j)$.
3) The adversary submits n, a message m, a set of router names $(P_1, \ldots, P_{n+1})$, an index j, and n key pairs $1 \leq i \leq n+1, i \neq j, (PK_i, SK_i)$.
4) The challenger checks that the router names are valid, that the public keys correspond to the secret keys and if so, sets $PK_j = PK$ and sets bit b at random.
5) If the adversary input was valid, the challenger picks $m' \leftarrow^R \mathcal{M}$ randomly and calculates: $(O_1, \ldots, O_j, \ldots, O_{n+1}) \leftarrow \text{FormOnion}(m, (P_1, \ldots, P_{n+1}), (PK_1, \ldots, PK_{n+1}))$

$(\bar{O}_1, \ldots, \bar{O}_j) \leftarrow$
$\mathrm{FormOnion}(m', (P_1, \ldots, P_j), (PK_1, \ldots, PK_j))$

6) • If $b = 0$, the challenger returns O_1 to the adversary.
   • Otherwise, the challenger returns $\bar{O}_1$ to the adversary.
7) The adversary may again query the oracle and submit any number of onions $O_i \neq O_j$, $O_i \neq \bar{O}_j$ of her choice to the challenger, to obtain the output of $\mathrm{ProcOnion}(SK, O_i, P_j)$.
8) The adversary then produces a guess b'.

Onion-Security is achieved if any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, cannot guess $b' = b$ with a probability non-negligibly better than $\frac{1}{2}$.

Onion-Security hence aims at guaranteeing that an adversary observing an onion before it is processed by an honest relay cannot discover information about the message it contains, or the path it subsequently takes. As the adversary controls all links, she could link message and receiver to the sender, otherwise. Further, step 7 provides protection against active modification attacks, as it allows processing of any modified onion.

The property however does not consider a malicious receiver or exit node, which hence might be able to discover information about the path or sender. Notice that this is exactly what happens in the attack on Sphinx; a malicious exit node learns information (the length) of the path.

C. Security against Malicous Receivers

In this subsection, we show the first shortcoming, missing protection against a malicious receiver, by giving a simplified broken protocol that allows the receiver to learn the complete path and yet achieves all suggested properties. Based on this discovery we introduce an improved property.

1) Insecurity: Signaling the Path: We first generalize the attack on Sphinx from Section IV-A, which only leaked the path length. As generalization we give a protocol that complies to the properties, but includes the complete path (including the sender) in the message. Thus, an adversarial receiver learns the complete path the onion took.

This weakness differs from the common assumption that one cannot protect senders that reveal their identity in their self-chosen message: independent of the message the sender chooses, the protocol always adds the complete sender-chosen path to it. Thus, an adversarial receiver always learns the sender and all relays independent of the sender's choice. Clearly, such an OR scheme should not be considered secure and private and hence should not achieve the OR properties.

a) Insecure Protocol 1: The main idea of this counterexample is to use a secure OR scheme and adapt it such that the path is part of the sent message.

More formally, our extended protocol $\Pi_{broken1}$ using $\mathrm{FormOnion}_{broken1}$ and $\mathrm{ProcOnion}_{broken1}$ is created from the "secure" onion routing protocol Π from [8]. Π transfers a message m from a sender P_0 to a receiver P_{n+1} over n intermediate routers $\{P_i\}$ for $1 \leq i \leq n$ using $\mathrm{FormOnion}_{\Pi}$ and $\mathrm{ProcOnion}_{\Pi}$.

Sender [$\mathrm{FormOnion}_{broken1}$]: The sender P_0 wants to send message $m \in \{0,1\}^{l_m - l_P}$ over path $\mathcal{P}$, where l_m is the length of messages in Π and l_P is the maximal length of the encoding of any valid path including the sender. $\mathrm{FormOnion}_{broken1}$ creates a new message $m' = m \| e(P_0 \| \mathcal{P})$, where e encodes the path and is padded to length l_P. $\mathrm{FormOnion}_{broken1}$ runs the original algorithm $\mathrm{FormOnion}_{\Pi}$ with the inputs chosen by the sender except that the message is replaced with m'.

Intermediate Router [$\mathrm{ProcOnion}_{broken1}$]: Any intermediate runs $\mathrm{ProcOnion}_{\Pi}$ on O_i to create O_{i+1} and sends it to the next router.

Receiver [$\mathrm{ProcOnion}_{broken1}$]: The receiver getting O_{n+1} executes $\mathrm{ProcOnion}_{\Pi}$ on it to retrieve m'. It learns the path from the last l_P bits and outputs the first $l_m - l_P$ bits as the received message.

b) Analysis regarding properties: The properties follow from the corresponding properties of the original protocol. As we only add and remove $e(P_0 \| \mathcal{P})$ to and from the message, the same path is taken and the complete onion layers O_i are calculated as before. Hence, *Correctness* and *Onion-Integrity* hold, and *re-wrapping* them is as difficult as before. Only *Onion-Security* remains. As Π has Onion-Security, the adversary cannot learn enough about the message included in the first onion layers to distinguish it from a random message. Thus, she especially cannot distinguish the last l_P bits from random ones in Π. As in Onion-Security the adversary learns nothing else, the adversary in $\Pi_{broken1}$ cannot distinguish our adapted message bits from random ones. Thus, adapting does not introduce any advantage in breaking Onion-Security.

2) Improved Property: Tail-Indistinguishability TI against a corrupted receiver : We construct the new property *Tail-Indistinguishability TI* to deal with malicious receivers. Therefore, the adversary has to get access to the onion layers after the last honest relay has processed them because a malicious receiver learns those. Our property challenges the adversary behind the last honest relay to distinguish between the onion generated with her original inputs, and a second onion that carries the identical message and follows the identical path behind the honest relay but otherwise was constructed with randomly chosen input, i.e. the path before the honest node is chosen randomly.

Note that this new property indeed prevents the insecurity given in Section IV-C1 and the attack on Sphinx: If the receiver is able to reconstruct any information of the path before the honest node, the adversary can compare this information with her input choice. In case the information does not match her choice, she knows that it must have been the second onion and thus is able to distinguish the onions.

Intuitively, the steps are the same as in Onion-Security described in Section IV-B3a, except that we change the answer to the challenge. This time we protect the last part of the path and output those layers. Since the receiver is corrupted, the message is learned by the adversary anyways and hence we use the same message for the alternative layers ($b = 1$).

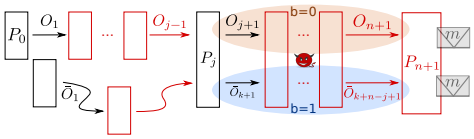

Fig. 3. Cases of TI illustrated: Red boxes are adversarial routers; black boxes honest and curvy arrows symbolize a random path possibly through many other adversarial routers. In case $b = 0$ the adversary chosen onion is observed at the end of the path (orange ellipse). For $b = 1$ onion layers that take the same path between P_j and P_{n+1} and include the same message (the blue ellipse), but differ otherwise, are observed instead. Earlier layers (before P_j) are in both cases not given to the adversary.

We illustrate the new outputs to the adversary in Fig. 3 and formally define the new property in our Definition 3.

Thus, our first new property TI is defined as:

Definition 3 (Tail-Indistinguishability TI):

1) The adversary receives as input the challenge public key PK, chosen by the challenger by letting $(PK, SK) \leftarrow G(1^\lambda, p, P_j)$, and the router name P_j.
2) The adversary may submit any number of onions O_i of her choice to the challenger. The challenger sends the output of $\text{ProcOnion}(SK, O_i, P_j)$ to the adversary.
3) The adversary submits a message m, a path $\mathcal{P} = (P_1, \ldots, P_j, \ldots, P_{n+1})$ with the honest node at position j, $1 \leq j \leq n+1$ of her choice and key pairs for all nodes (PK_i, SK_i) ($1 \leq i \leq n+1$ for the nodes on the path and $n+1 < i$ for the other relays).
4) The challenger checks that the router names are valid, that the public keys correspond to the secret keys and that the same key pair is chosen if the router names are equal, and if so, sets $PK_j = PK$ and sets bit b at random.
5) The challenger creates the onion with the adversary's input choice:

   $$(O_1, \ldots, O_{n+1}) \leftarrow \text{FormOnion}(m, \mathcal{P}, (PK)_{\mathcal{P}})$$

   and a random onion with a randomly chosen path $\bar{\mathcal{P}} = (\bar{P}_1, \ldots, \bar{P}_k = P_j, \ldots, \bar{P}_{\bar{n}+1} = P_{n+1})$, that includes the subpath from the honest relay to the corrupted receiver starting at position k ending at $\bar{n}+1$:

   $$(\bar{O}_1, \ldots, \bar{O}_{\bar{n}+1}) \leftarrow \text{FormOnion}(m, \bar{\mathcal{P}}, (PK)_{\bar{\mathcal{P}}})$$

6) • If $b = 0$, the challenger gives (O_{j+1}, P_{j+1}) to the adversary
   • Otherwise, the challenger gives $(\bar{O}_{k+1}, \bar{P}_{k+1})$ to the adversary
7) The adversary may submit any number of onions O_i of her choice to the challenger. The challenger sends the output of $\text{ProcOnion}(SK, O_i, P_j)$ to the adversary.
8) The adversary produces guess b' .

TI is achieved if any PPT adversary $\mathcal{A}$, cannot guess $b' = b$ with a probability non-negligibly better than $\frac{1}{2}$.

D. Linking Protection

The flaw of the previous section is not the only one the proposed properties missed. Here, we introduce a second insecure protocol, which allows to bypass honest nodes by linking onions, and construct a new property against this weakness.

1) Insecurity: Including Unique Identifiers: We show that reidentifying the same onion after processing at a honest node is not prevented by the original properties with the following obviously insecure protocol.

a) Insecure Protocol 2: The protocol $\Pi_{broken2}$ when creating an onion independently draws a random identifier ID and appends it to each layer of the created onion. The ID makes the onion easily traceable, as it remains identical throughout the processing of the onion at any relay. For the proof of the properties we need to construct an extension that prevents modification of the ID. We provide the details of the extension and broken scheme in the extended version of this paper [24].

b) Analysis regarding Properties: Without going into the details here, we note that none of the properties protects from embedded identifiers, which are identical for all onion layers of the same onion, but different for other onions: *Integrity*, *Correctness* and *Wrap-Resistance* are not influenced by this adaptation as the same path is taken, the onion layers (without appended ID) are constructed as before and thus are as hard to re-wrap as before. *Onion-Security* is not broken as the extension protects against modification of both the appended ID and extension, and thus calling the oracle with a modified ID or extension is useless. Finally, the appended ID does not include any information about the input used to form the onion and hence does not help to distinguish the onion with inputs of the adversary from another onion.

Note that also Tail-Indistinguishability cannot protect against linking as only one onion layer is given to the adversary.

2) Improved Property: Layer-Unlinkability LU against bypassing honest nodes: To explicitly model that output onions shall not be linkable to the corresponding inputs of the relays, the adversary has to get onion layers both before and after they are processed at the honest relay. Our property challenges the adversary observing an onion going from the sender to an honest relay, to distinguish between the onion generated with her original inputs O, and a second onion $\bar{O}$. The path of the alternative onion $\bar{O}$ includes the original path from the sender to the honest node, but all other parameters are chosen randomly. Thus, there might be a path before the sender node and both the path after the honest node and the message can differ. Additionally, the adversary always observes the onion generated by processing O at the honest relay. We illustrate the new challenge outputs in Fig. 4.

Note that this new property indeed prevents the insecurity given in Section IV-D1: If the adversary can decide that the provided onions belong together, she knows that the original onion has been sent and thus she is able to distinguish the onions.

We again adapt the original Onion-Security explained in Section IV-B3a with the difference that the adversary now gets the layers of O after the honest relay and either O's layers between the honest sender and relay or $\bar{O}$'s layers in Step 6. This is our new property LU, which is formally defined in Def. 4.

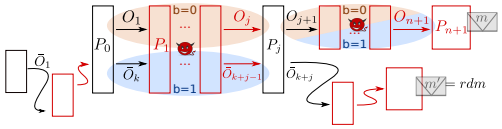

Fig. 4. Cases of LU illustrated: Red boxes are corrupted routers, black honest routers and curved arrows represent randomly chosen paths. In case $b = 0$ the adversary chosen onion, sent from P_0, is observed on the complete path $\mathcal{P}$ starting from when it arrives at the first relay P_1. For $b = 1$ the onion layers in the first orange ellipse are replaced with those of a randomly drawn onion, that take the same path between P_0 and P_j (the blue ellipse), but differ otherwise and might have traveled from another honest sender to P_0 earlier.

Definition 4 (Layer-Unlinkability LU):

1) – 4) as in Def. 3
5) The challenger creates the onion with the adversary's input choice:

$$(O_1, \ldots, O_{n+1}) \leftarrow \text{FormOnion}(m, \mathcal{P}, (PK)_{\mathcal{P}})$$

and a random onion with a randomly chosen path $\bar{\mathcal{P}} = (\bar{P}_1, \ldots, \bar{P}_k = P_1, \ldots, \bar{P}_{k+j} = P_j, \bar{P}_{k+j+1}, \ldots, \bar{P}_{\bar{n}+1})$, that includes the subpath from the honest sender to honest node of $\mathcal{P}$ starting at position k ending at $k+j$ (with $1 \leq j+k \leq \bar{n}+1 \leq N$), and a random message $m' \in \mathcal{M}$:

$$(\bar{O}_1, \ldots, \bar{O}_{\bar{n}+1}) \leftarrow \text{FormOnion}(m', \bar{\mathcal{P}}, (PK)_{\bar{\mathcal{P}}})$$

6) • If $b = 0$, the challenger gives $(O_1, \text{ProcOnion}(O_j))$ to the adversary.
• Otherwise, the challenger gives $(\bar{O}_k, \text{ProcOnion}(O_j))$ to the adversary.
7) The adversary may submit any number of onions O_i, $O_i \neq O_j$, $O_i \neq \bar{O}_{k+j}$ of her choice to the challenger. The challenger sends the output of $\text{ProcOnion}(SK, O_i, P_j)$ to the adversary.
8) The adversary produces guess b' .

LU is achieved if any PPT adversary $\mathcal{A}$, cannot guess $b' = b$ with a probability non-negligibly better than $\frac{1}{2}$.

E. Improved Properties imply Ideal Functionality

In this section we first informally argue and then formally prove that our two new properties, together with Onion-Correctness, are sufficient for the ideal functionality. For easier discussion, we summarize the different outputs of the security properties in Fig. 5.

Informally: In case of sender corruption, the ideal functionality outputs all information given as input to FormOnion, and hence we do not need to provide any protection in this case.

Considering honest senders, the ideal functionality outputs only the path sections introduced by cutting at honest nodes

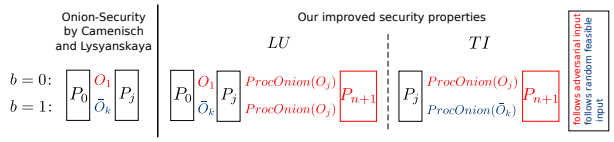

Fig. 5. Difference in security properties illustrated: While in original Onion-Security no processed onion after P_j is output, LU outputs the processing and TI challenges to distinguish it from randomness.

together with random onion IDs or if the receiver is compromised, additionally the message. These IDs are independently drawn for each such section and thus cannot be linked.

The idea to show the same privacy for communications with honest senders is simple: for every path section instead of the original onion layers we give the adversary without her noticing it layers of a random replacement onion. The replacement onion only corresponds with the original onion in characteristics she also learns about the onion in the ideal functionality. Namely those characteristics are the path sections and if the receiver is corrupted, the message. The replacements can obviously not be linked or leak any other information as all their (other) parameters have been chosen randomly.

Our properties are sufficient to show that the adversary cannot notice the replacement: LU allows to replace any onion layers on a path section between two honest nodes with onion layers that are (except for the fact that they use the same path section) chosen completely at random. The adversary is not able to notice the replacement as she cannot distinguish the onion layers in the LU game. This allows us to replace all layers in communications between honest senders and receivers, and all except the last section in communications between honest senders and corrupted receivers.

For replacement on the last part of a path with a corrupted receiver we need our other property TI. TI allows to replace any onion layers on a path section between an honest node and a corrupted receiver with onion layers that are (except for the fact that they use the same path section and carry the same message) chosen completely random. The adversary is not able to notice the replacement as she cannot distinguish the onion layers in the TI game. This completes our informal argument.

Formally: Similar to Camenisch and Lysyanskaya we assume a secure protocol to distribute public keys. We consider key distribution outside of the scope of this paper.

We now show that our new security properties are indeed sufficient to realize the ideal functionality. Therefore, we define a secure OR scheme to fulfill all our properties:

Definition 5: A secure OR scheme is a triple of polynomial-time algorithms $(G, FormOnion, ProcOnion)$ as described in Section II-E2 that achieves Onion-Correctness (Definition 1), Tail-Indistinguishability (Definition 3), as well as Layer-Unlinkability (Definition 4).

Following Camenisch and Lysyanskaya, we build a protocol from any secure OR scheme:

Definition 6: OR protocol Π is a secure OR protocol if it is based on a secure OR scheme $(G, \text{FormOnion}, \text{ProcOnion})$

and works as follows:

Setup: Each node P_i generates a key pair[9] $(SK_i, PK_i) \leftarrow G(1^\lambda, p, P_i)$ and publishes PK_i.

Sending a Message: If P_S wants to send $m \in \mathcal{M}$ to P_r over path $P_1, \ldots, P_n$ with $n < N$, he calculates $(O_1, \ldots, O_{n+1}) \leftarrow \mathrm{FormOnion}(m, (P_1, \ldots, P_n, P_r), (PK_1, \ldots, PK_n, PK_r))$ and sends O_1 to P_1.

Processing an Onion: P_i received O_i and runs $(O_j, P_j) \leftarrow \mathrm{ProcOnion}(SK_i, O_i, P_i)$. If $P_j = \perp$, P_i outputs "Received $m = O_j$" in case $O_j \neq \perp$ and reports a fail if $O_j = \perp$. Otherwise P_j is a valid relay name and P_i generates a random $temp$ and stores $(temp, (O_j, P_j))$ in its outgoing buffer and notifies the environment about $temp$.

Sending an Onion: When the environment instructs P_i to forward $temp$, P_i looks up $temp$ in its buffer. If P_i does not find such an entry, it aborts. Otherwise, it found $(temp, (O_j, P_j))$ and sends O_j to P_j.

To show that any secure OR protocol Π realizes the ideal functionality, we prove that any attack on the secure OR protocol can be simulated in the ideal functionality. As the simulator only gets the outputs of the ideal functionality and thus no real onions, it simulates them with the closest match it can create: replacement onions that take the same path (and, if sent to corrupted receivers, include the same message). Due to our new security properties, we know that such a replacement cannot be distinguished. The full proof is included in Appendix B.

Theorem 2: A secure OR protocol following Definition 6 UC-realizes the ideal functionality $\mathcal{F}$.

V. Second Pitfall: Undervalued Oracles

We discovered a new attack on HORNET whose existence cannot be explained with the shortcomings of the properties of Camenisch and Lysyanskaya. The reason for this attack is not in the properties used for the proof, but in how the properties are proven. It turns out that on many occasions the properties have not been proven correctly; more precisely the oracles have been wrongly adapted or ignored.

We start this section by describing our attack, then explain how the oracles have been modified and how the correct use of oracles detects the attack.

A. Attacking HORNET's Data Transmission

HORNET was proposed as a network level anonymity system for the anonymized transmission of arbitrary higher layer packets. The latter can be expected to match specific formats or contain interpretable content, e.g. natural language. Hence the receiver can very likely distinguish real messages from random bit strings of the same length in HORNET's transmission phase.

HORNET uses a pseudo-random permutation (PRP) in CBC mode[10] to form layered encryption of its payload, but does not implement integrity checks at the processing relays for it.

[9]We assume the PK_i are checked to be well formed as part of the key distribution mechanism that is outside the scope of this work.

[10]Note, that the paper is not entirely clear about this point, as it states that HORNET uses a "stream-cipher", which would make our attack stronger, "in CBC mode", suggesting that instead they actually use a PRP.

TABLE I
Observable linkings on different systems; (✓) if attack works only under violation of the adversary model

System	Sender-Message	Sender-Receiver	Sender-Exit node
Improved Minx		✓	✓
Sphinx (receiver ≠ exit node)			✓
Sphinx (receiver = exit node)[13]		✓	✓
HORNET (Setup)		✓	✓
TARANET (Setup)		(✓)	(✓)
HORNET (Data)	✓	✓	✓
TARANET (Data)			

An attacker that controls the first relay[11] and the receiver can link sender and receiver (thus break relationship anonymity $S\overline{R\!L}$) and this adversary can also link large parts of the message to its sender (break sender anonymity $S\overline{M\!L}$) with the following attack:

1) The adversary flips bits in the last k blocks of the data payload of the HORNET packet sent from the sender.
2) The packet is sent through the chain of relays as intended because the header was not modified and the payload's integrity is not protected. The payload is decrypted using the block cipher in CBC mode.
3) The receiver checks the last k blocks. They either contain random bits (i.e. the sender was communicating with this receiver and the preceding decrypted blocks contain parts of the original message) or it conforms to a real message (i.e. the sender was not communicating with this receiver).

Varying k the success of distinguishing real messages from some ending with random bit strings can be adjusted at the cost of learning less about the real message.

The described attack lies well within the adversary model of HORNET: it allows a fraction of nodes to be actively controlled by the adversary and aims at sender anonymity, even if the receiver is compromised, and relationship anonymity, even if one of the end hosts is compromised.

Further, the attack can be varied to link senders and receivers in the improved Minx or, if it is used with an unintended addressing model like in HORNET's or TARANET's setup phase[12], in Sphinx (See Table I for a summary. For more information on how to vary the attack, we refer the interested reader to our extended version [24]).

B. Mistake in the Proofs

The shared pitfall are the oracles. In HORNET's analysis this attack was excluded as the oracles were not taken into account. The proof of TARANET ignores the oracles as well, yet its transmission phase incidentally protects against our attack. Sphinx, the improved Minx and even an extension in [8] restrict the oracle in our Step 7 to only allow non-duplicate onions, i.e. those with a changed header. This weakens the properties too much, as the limited oracle incidentally loses protection

[11]Technically, controlling the link from the sender to the first relay is enough. However, whether the adversary controls links is not explicitly stated in [11].

[12]Although this attack works for TARANET, it is outside TARANET's attacker model as the receiver needs to be corrupted.

[13]We stress that this model was never intended by Sphinx, but other works used Sphinx that way.

from modification attacks, where the onion is modified before it ever reached the honest node.

Note, that our property LU (and even the insecure original Onion-Security) indeed cannot be fulfilled if the before mentioned attack (Section V-A) works: The adversary alters only the payload of the challenge onion and queries the oracle with the modified onion. As processing at the honest node is not aborted for modified onions, the adversary learns the next relay after the honest node. She can thus decide whether the next relay corresponds to her choice ($b = 0$) or not ($b = 1$).

We want to stress that this is not the only attack that prevents HORNET from achieving LU. Another exploits the usage of sessions (more in Section VII-C2).

VI. Proving the Adapted Sphinx secure

Sphinx specifies to use a header and a payload. The original Sphinx [13] suggests per-hop integrity protection only for the header as an integrity check for the payload conflicts with their support for replies. Thus, as mentioned in Section V-A Sphinx allows to link sender and exit node. As this linking is not possible in the ideal functionality, Sphinx, even with the flaw from Section IV-A fixed, cannot realize the ideal functionality.

Beato et al. however proposed an adaptation to Sphinx, to simplify the protocol and improve security and performance at the cost of losing support for replies [5]. Thereby, they introduce integrity checks of the payload at each hop. As this prevents the linking attack, we decided to analyze this version of Sphinx, adapted with the small fix to the attack from Section IV-A known from the Sphinx implementation, for compliance with our properties for secure OR protocols. Note, that in compliance to Beato et al. this variation covers only the forward phase and no replies.

The proof for Onion-Correctness follows the ideas in [13]. To analyze LU and TI, we successively define games with marginally weaker adversary models. Arguing how each step follows from reasonable assumptions, we terminally reduce it to the security of an authenticated encryption scheme and the DDH assumption. We provide the detailed proof in Appendix C, and it leads to the following theorem:

Theorem 3: Beato's Sphinx variation, adapted with the fix to the attack from Section IV-A, is a secure OR scheme.

As this implies that it realizes the ideal functionality, we can conclude that it achieves confidentiality ($M\overline{O}$) for honest senders with honest receivers, and sender ($SM\overline{L}$) and relationship anonymity ($SR\overline{L}$) for honest senders with corrupted receivers. This holds for a restricted adversary model, which does not allow timing attacks or attacks that lead to the dropping of onions. This limitation conforms to the adversary model of the original Sphinx, which is used in the adapted version as well.

VII. Discussion

In this section, we relate our properties to known attacks and give further comments about the limitations of using them.

A. Onion-Security Properties vs. Existing OR Attacks

Our new properties prevent well-known attacks on OR if they comply to the adversary model of the ideal functionality. Passive *linking attacks* e.g. based on length of the onion layer, or the length of the included message are prevented (attacks on LU would otherwise be possible). Additionally, our properties imply non-deterministic encryption in $\mathrm{FormOnion}$, as the adversary could use $\mathrm{FormOnion}$ on its chosen parameters and compare the results, otherwise.

In *tagging attacks* the attacker modifies an onion and recognizes it later based on the modification. To be useful, the tagging has to preserve some information of the original communication, e.g. a part of the path or the message. This translates to an attack on LU that uses an oracle to learn the output of a tagged challenge onion after processing at an honest relay, and deciding if it relates to the chosen input ($b = 0$), or not.

Duplicate attacks assume an adversary that is able to create an onion that equals an intercepted onion in parts of the input, e.g. the message, that can later be observed, but is not bit-identical. Such onions appear different at the relays and hence may not be detected by duplicate protection. They still jeopardize anonymity, as the adversary may notice their repeated delivery to the receiver. Our properties protect from duplicate attacks, as an adversary that was able to create a duplicate onion breaks LU by learning the message or path contained in the challenge onion by using the oracle.

Replay attacks (duplicate attacks with bit-identical onion) are possible in the ideal functionality and consequently not necessarily prevented.

The *n-1 Attack*, where all but one onion is known to the adversary, and hence the remaining one can be traced, is possible in the ideal functionality and thus not mitigated by the properties.

B. Adapting Our Properties

There are cases, in which our properties need adaptation:

Correctness: Due to practical reasons, space-efficient data structures like Bloom filters are frequently used for duplicate detection. Bloom filters exhibit false-positive detections (that is non-duplicate packets are detected as duplicates with a certain probability), but no false-negatives (duplicates are always detected). However, the false-positive probability of a Bloom filter depends on its configuration and is usually not negligible. This can be covered by extending our Onion-Correctness to δ-Onion-Correctness, thus accepting a correctness failure at a probability of at most δ.

Security properties and Cascades: So far we assumed that the replacement onion is any onion that shares the observed part of the path. This naturally applies for free routing protocols, in which the sender randomly picks any path, and which is considered by the ideal functionality. When analyzing OR with fixed cascades, some adaptations are necessary. Adaptation and changes in the analysis for the adapted ideal functionality, however, are straightforward: senders can only choose a cascade instead of a path. This results in a different path choice in the

adversary class and thus in a slightly different anonymity set. In the game, the path of the replacement onion finally has to match the cascade of the challenge onion (this can be assured in Step 5 of both LU and TI).

C. Limitations

As limitations of this paper, we recall the adversary model, the anonymity set, and discuss the limits inherited from the ideal functionality.

1) Adversary Model and Anonymity Set: We fully assumed the adversary model of Camenisch and Lysyanskaya. This adversary model does not allow for traffic analysis as timing information is removed and no delaying or dropping is allowed by the adversary. Although this adversary model does not seem very realistic, the analysis is useful to split the proof. Upon showing the protocol's privacy for the restricted adversary model of the ideal functionality by proving the properties, only the privacy for the remaining attacks has to be shown.

We restrict the paths in the adversary class to include at least one honest relay to achieve the notions. This means that the anonymity set consists only of the users whose onions share an honest relay and are processed together.

2) Reply Channels and Sessions: All systems that proved privacy with the properties consider a reply channel, for example to respond to an anonymous sender. None, however, analyzes the backward phase separately. They only show indistinguishability to the forward onions (if at all), implying that the same security properties are used for the reply channel. However, our analysis showed that the privacy goals except confidentiality ($M\overline{O}$) are only guaranteed for an honest sender. In a reply phase this sender is the original receiver, which cannot ultimately be considered honest. Thus, proving the properties does not guarantee the anonymity of the initial sender for a corrupted receiver in the reply phase.

HORNET and TARANET additionally introduce sessions. Their data transmission phase reuses the same path and header to efficiently send multiple onions. The ideal functionality does not cover sessions. As for a corrupted relay it is always possible to link onions of the same session, neither the properties, nor ultimately the ideal functionality can be shown in this case.

Besides noticing this insufficiency, sending replies to the sender or using sessions is outside of the scope of this paper. We conjecture that both issues can be solved in future work by changing the ideal functionality and introducing additional properties. For this paper, we deemed it however more important to explain and correct all mistakes related to the simple sending with OR in detail.

D. Some Thoughts about Mix Networks

Mix networks in addition to onion processing include reordering of onions (usually by delaying them for some time), to conceal timing information and prevent linking outgoing to incoming onions based on their order and timing. The ideal functionality, as well as both the original and our properties all do not consider timing attacks. Although none of the widely deployed anonymization systems considers this, a real anonymous communication network of course should prevent linking based on timings. From the perspective of this work we consider this an extension, as all properties presented here need to be met by mix networks, as well, to prevent linking based on the onions and their processing at honest nodes.

E. Extended Version

Our extended version of this paper [24] contains technical details we excluded for this version due to space limitations. These comprise the technical proofs of the notions the ideal functionality does and does not achieve (including attacks for stronger notions), a scheme and the corresponding proofs illustrating the second insecurity (Section IV-D1) of the properties from [8] and the proof that Wrap-Resistance and Onion-Integrity of [8] do not need to be proven for privacy reasons.

VIII. Conclusion and Future Work

Camenisch and Lysyanskaya have made a seminal attempt to formally analyze the predominant anonymization approach of OR in [8]: They design an ideal functionality for OR in the UC model and suggest properties to analyze protocols and real-world systems. A whole family of subsequent OR schemes based their security analyses on this work.

Analyzing approaches from this family, we discovered a new, severe vulnerability and explained one that was known. We presented a new attack to completely break sender and relationship anonymity in HORNET. Further as known and corrected in the implementation, in Sphinx as in [13] the anonymity set can be reduced by discovering the used path length.

As these attacks contradict the proofs in the respective papers, we set out to formally analyze the used proof strategy proposed in [8]. First, we confirmed that the foundation of the proof, the ideal functionality, indeed guarantees privacy.

Second, we explained the reason for the attack on Sphinx: the properties as originally suggested by Camenisch and Lysyanskaya are insufficient. To resolve this situation, we fixed one property, developed two new properties, and proved that achieving these three properties implies the privacy of the ideal functionality: sender anonymity and relationship anonymity against corrupted receivers in an adversary model that limits onion dropping and timing-based attacks.

Third, we explained the reason for the attack on HORNET: the original Onion-Security property would have prevented it, but has been proven incorrectly. Proving a variation of Sphinx secure, we demonstrated how systems can be analyzed using our new properties.

We wish to point out that several of the published systems consider reply channels as well as sessions – which indeed are not covered by the ideal functionality of [8]. Therefore, much is left to be done: while we repaired the anonymization for the simple delivery of a message from a sender to a receiver, modeling reply channels and sessions is left for future work. Further, analyses and proofs for the security and privacy of other onion routing protocols beyond the variation of Sphinx need to be conducted, by using our or adapted properties.

ACKNOWLEDGMENT

We thank our shepherd Ian Goldberg and the anonymous reviewers for their very valuable feedback. This work in part was funded by DFG EXC 2050/1 – ID 390696704.

REFERENCES

[1] E. D. Ayele. Analysis and deployment of the BitTorrent protocol for Community Ad-hoc Networks. Technical report, TU Delft, 2011.
[2] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi. Anoa: A framework for analyzing anonymous communication protocols. *Journal of Privacy and Confidentiality*, 2017.
[3] M. Backes, P. Manoharan, and E. Mohammadi. Tuc: Time-sensitive and modular analysis of anonymous communication. In *IEEE CSF*, 2014.
[4] E. Balkovich, D. Prosnitz, A. Boustead, and S. C. Isley. *Electronic Surveillance of Mobile Devices*. Rand Corporation, 2015.
[5] F. Beato, K. Halunen, and B. Mennink. Improving the sphinx mix network. In *Cryptology and Network Security*, 2016.
[6] R. Berman, A. Fiat, M. Gomulkiewicz, M. Klonowski, M. Kutylowski, T. Levinboim, and A. Ta-Shma. Provable unlinkability against traffic analysis with low message overhead. *Journal of Cryptology*, 2015.
[7] J.-M. Bohli and A. Pashalidis. Relations among privacy notions. *ACM TISSEC*, 2011.
[8] J. Camenisch and A. Lysyanskaya. A formal treatment of onion routing. In *Annual International Cryptology Conference*, 2005.
[9] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE FOCS*, 2001.
[10] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.
[11] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. HORNET: High-speed onion routing at the network layer. In *ACM CCS*, 2015.
[12] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso. TARANET: Traffic-Analysis Resistant Anonymity at the NETwork layer. *IEEE EuroS&P*, 2018.
[13] G. Danezis and I. Goldberg. Sphinx: A compact and provably secure mix format. In *IEEE S&P*, 2009.
[14] G. Danezis and B. Laurie. Minx: A simple and efficient anonymous packet format. In *WPES*, 2004.
[15] J. P. Degabriele and M. Stam. Untagging Tor: a formal treatment of onion encryption. In *Theory and Applications of Cryptographic Techniques*, 2018.
[16] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.
[17] J. Feigenbaum, A. Johnson, and P. Syverson. A model of onion routing with provable anonymity. In *Financial Cryptography and Data Security*, 2007.
[18] J. Feigenbaum, A. Johnson, and P. Syverson. Anonymity analysis of onion routing in the universally composable framework. In *2012 Workshop on Provable Privacy*, 2012.
[19] A. Fujioka, Y. Okamoto, and T. Saito. Security of sequential multiple encryption. In *International Conference on Cryptology and Information Security in Latin America*, 2010.
[20] N. Gelernter and A. Herzberg. On the limits of provable anonymity. In *ACM WPES*, 2013.
[21] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. In *International workshop on information hiding*, 1996.
[22] A. Hevia and D. Micciancio. An indistinguishability-based characterization of anonymous channels. *Lecture Notes in Computer Science*, 2008.
[23] C. Kuhn, M. Beck, S. Schiffner, E. Jorswieck, and T. Strufe. On privacy notions in anonymous communication. *PoPETs*, 2019.
[24] C. Kuhn, M. Beck, and T. Strufe. Breaking and (Partially) Fixing Provably Secure Onion Routing. *arXiv e-prints*, page arXiv:1910.13772, 2019.
[25] S. Mauw, J. H. Verschuren, and E. P. de Vink. A formalization of anonymity and onion routing. In *European Symposium on Research in Computer Security*, 2004.
[26] K. Peng. A general and efficient countermeasure to relation attacks in mix-based e-voting. *Int. J. Inf. Secur.*, 10(1), Feb. 2011.
[27] D. J. Pohly and P. McDaniel. Modeling Privacy and Tradeoffs in Multichannel Secret Sharing Protocols. In *IEEE/IFIP DSN*, 2016.
[28] P. H. Potgieter. An introduction to new media for South African students. 2009.
[29] E. Shimshock, M. Staats, and N. Hopper. Breaking and provably fixing minx. In *PETS*, 2008.
[30] F. Tschorsch. *Onions in the Queue: An Integral Networking Perspective on Anonymous Communication Systems*. PhD thesis, Humboldt-Universitt zu Berlin, 2016.

APPENDIX

A. Definition Privacy Notions

1) Game: The model uses $r = (u, u', m, aux)$ to denote the communication of message m from sender u to receiver u' with auxiliary information aux. Communications that are processed together are grouped in batches $\underline{r}$. The adversary decides on two scenarios. Those are a sequence of pairs of batches. The challenger verifies every pair of batches $\underline{r}_0, \underline{r}_1$ regarding the analyzed privacy notion, i.e. they differ only in private information. If the check succeeds, the challenger picks a random b and simulates the protocol for the corresponding batch $\underline{r}_b$. The adversary can issue more batches and finally makes a guess g for b. If the adversary cannot guess $g = b$ correctly with a more than negligibly better probability than $\frac{1}{2}$, the notion is achieved as nothing private can be learned.

2) Important Notions: We always consider the checked batches $\underline{r}_0, \underline{r}_1$, which for $b \in \{0, 1\}$ include the communications $r_{bj} = (u_{b_j}, u'_{b_j}, m_{b_j}, aux_{b_j})$ with $j \in \{1, \dots l\}$.

For $M\overline{O}$ two batches may only differ in the messages:

Definition 7 ($M\overline{O}$ i. a. w. [23]): The batches are valid for $M\overline{O}$, iff for all $j \in \{1, \dots l\}$: $r_{1j} = (u_{0_j}, u'_{0_j}, \mathbf{m_{1_j}}, aux_{0_j})$.

For $SM\overline{L}$ only the senders may differ and further each sender has to send the same number of messages in the two batches. To define this, [23] formally defines Q_b. Here we use a less formal description: $Q_b := \{(u, n) \mid u \text{ sends } n \text{ messages in } \underline{r}_b\}$.

Definition 8 ($SM\overline{L}$ i. a. w. [23]): The batches are valid for $SM\overline{L}$, iff for all $j \in \{1, \dots l\} : r_{1j} = (\mathbf{u_{1_j}}, u'_{0_j}, m_{0_j}, aux_{0_j})$ and $Q_0 = Q_1$.

$RM\overline{L}$ is similar, but for receivers: $Q'_b := \{(u', n) \mid u' \text{ receives } n \text{ messages in } \underline{r}_b\}$.

Definition 9 ($RM\overline{L}$ i. a. w. [23]): The batches are valid for $RM\overline{L}$, iff for all $j \in \{1, \dots l\} : r_{1j} = (u_{0_j}, \mathbf{u'_{1_j}}, m_{0_j}, aux_{0_j})$ and $Q'_0 = Q'_1$.

M_{SR}	batch 0	batch 1
either this order	A → B	A → D
	C → D	C → B
or this order	C → D	C → B
	A → B	A → D

Fig. 6. Batches in M_{SR} illustrated

$SR\overline{L}$ allows only sender and receiver to differ and has the complex requirement M_{SR}. M_{SR} requires that the batches only differ in two senders (A, C) and two receivers (B, D). In the case $b = 0$: A must communicate with B, and C with D; in the case $b = 1$: A with D, and C with B. The order of the two communications in the batch is chosen randomly by the challenger. Before, between and after those communications

multiple communications that are equal in both batches can occur. The possible communications are depicted in Fig. 6.

Definition 10 ($SR\overline{L}$ i.a. w. [23]): The batches are valid for $SR\overline{L}$, iff for all $j \in \{1, \dots l\} : r_{1j} = (\mathbf{u_{1_j}}, \mathbf{u'_{1_j}}, m_{0_j}, aux_{0_j})$ and M_{SR}.

3) Corruption: User corruption is realized by returning internal information of the user (keys, current state etc.). X_{c^0} ensures that the adversary is not allowed corruption. The other corruption options add requirements for the two batches:

Definition 11 (Corruption): Let $\hat{U}$ be the set of all corrupted users. The following options are met, iff:

$$X_s : \forall (u, u', m, aux) \in \underline{r}_0 \cup \underline{r}_1 : u \notin \hat{U}$$
$$X_e : \forall \hat{u} \in \hat{U} : r_{0_i} = (\hat{u}, \_, m, \_) \implies r_{1_i} = (\hat{u}, \_, m, \_)$$
$$\wedge\, r_{0_i} = (\_, \hat{u}, m, \_) \implies r_{1_i} = (\_, \hat{u}, m, \_)$$

B. Proof of new Properties

Our proof follows in large parts the argumentation from [8]. For UC-realization, we show that every attack on the real world protocol Π can be simulated by an ideal world attack without the environment being able to distinguish those.

1) Constructing $\mathcal{S}$: $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}$ as the ideal world adversary, and simulates the real-world honest parties for the real world adversary $\mathcal{A}$. All outputs $\mathcal{A}$ does are forwarded to the environment by $\mathcal{S}$.

First, $\mathcal{S}$ carries out the trusted set-up stage: it generates public and private key pairs for all the real-world honest parties. $\mathcal{S}$ then sends the respective public keys to $\mathcal{A}$ and receives the real world corrupted parties public keys from $\mathcal{A}$.

The simulator $\mathcal{S}$ maintains two internal data structures:

- The r-list consisting of tuples of the form $(r_{temp}, nextRelay, temp)$. Each entry in this list corresponds to a stage in processing an onion that belongs to a communication of an honest sender. By stage, we mean that the next action to this onion is adversarial (i.e. it is sent over a link or processed by an adversarial router).
- The O-list containing onions sent by corrupted senders together with the information about the communication $(onion, nextRelay, information)$.

a) $\mathcal{S}$'s behavior on a message from $\mathcal{F}$: In case the received output belongs to an adversarial sender's communication[14]:

Case I: "start belongs to onion from P_S with $sid, P_r, m, n, \mathcal{P}$". This is just the result of $\mathcal{S}$s reaction to an onion from $\mathcal{A}$ that was not the protocol-conform processing of an honest sender's communication (Case VIII). $\mathcal{S}$ does nothing.

Case II: any output together with "$temp$ belongs to onion from P_S with $sid, P_r, m, n, \mathcal{P}$" for $temp \notin \{\text{start}, \text{end}\}$. This means an honest relay is done processing an onion received from $\mathcal{A}$ that was not the protocol-conform processing of an honest sender's communication (processing that follows Case VII). $\mathcal{S}$ finds $(onion, nextRelay, information)$ with this inputs as $information$ in the O-list (notice that there has to be such an entry) and sends the onion $onion$ to $nextRelay$ if it is an adversarial one, or it sends $onion$, as if it is transmitted, to the $\mathcal{A}$'s party representing the link between the currently processing honest relay and the honest $nextRelay$.

Case III: any output together with "end belongs to onion from P_S with $sid, P_r, m, n, \mathcal{P}$". This is just the result of $\mathcal{S}$'s reaction to an onion from $\mathcal{A}$. $\mathcal{S}$ does nothing.

In case the received output belongs to an honest sender's communication:

Case IV: "Onion $temp$ from P_{o_i} routed through () to $P_{o_{i+1}}$". In this case $\mathcal{S}$ needs to make it look as though an onion was passed from the honest party P_{o_i} to the honest party $P_{o_{i+1}}$: $\mathcal{S}$ picks pseudo-randomly (with $temp$ as seed) a path $\mathcal{P}_{rdm}$, of valid length that includes the sequence of P_{o_i} to $P_{o_{i+1}}$ starting at node j, and a message m_{rdm}. $\mathcal{S}$ calculates $(O_1, \dots, O_n) \leftarrow \text{FormOnion}(m_{rdm}, \mathcal{P}_{rdm}, (PK)_{\mathcal{P}_{rdm}})$ and sends the onion O_{j+1} to $\mathcal{A}$'s party representing the link between the honest relays as if it was sent from P_{o_i} to $P_{o_{i+1}}$. $\mathcal{S}$ stores (O_{j+1},$P_{o_{i+1}}$,$temp$) on the r-list.

Case V: "Onion $temp$ from P_{o_i} routed through $(P_{o_{i+1}}, \dots, P_{o_{j-1}})$ to P_{o_j}". $\mathcal{S}$ picks pseudo-randomly (with $temp$ as seed) a path $\mathcal{P}_{rdm}$ of valid length that includes the sequence of P_{o_i} to P_{o_j} starting at the k-th node and a message m_{rdm} and calculates $(O_1, \dots, O_n) \leftarrow \text{FormOnion}(m_{rdm}, \mathcal{P}_{rdm}, (PK)_{\mathcal{P}_{rdm}})$ and sends the onion O_{k+1} to $P_{o_{i+1}}$, as if it came from P_{o_i}. $\mathcal{S}$ stores $(O_{k+j-i}, P_{o_j}, temp)$ on the r-list.

Case VI: "Onion from P_{o_i} with message m for P_r routed through $(P_{o_{i+1}}, \dots, P_{o_n})$". $\mathcal{S}$ picks randomly a path $\mathcal{P}_{rdm}$ of valid length that includes the sequence of P_{o_i} to P_r at the end (staring at the k-th node) and calculates $(O_1, \dots, O_n) \leftarrow \text{FormOnion}(m_t, \mathcal{P}_{rdm}, (PK)_{\mathcal{P}_{rdm}})$ and sends the onion O_{k+1} to $P_{o_{i+1}}$, as if it came from P_{o_i}.

b) $\mathcal{S}$'s behavior on a message from $\mathcal{A}$: $\mathcal{S}$, as real world honest party P_i, received an onion O from $\mathcal{A}$ as adversarial player P_a.

Case VII: $(O, P_i, temp)$ is on the r-list for some $temp$. In this case O is the protocol-conform processing of an onion from a communication of an honest sender. $\mathcal{S}$ calculates $\text{ProcOnion}(SK(P_i), O, P_i)$. If it returns a fail ($O$ is a replay that is detected and dropped by Π), S does nothing. Otherwise, $\mathcal{S}$ sends the message (Deliver Message, $temp$) to $\mathcal{F}$.

Case VIII. $(O, P_i, temp)$ is not on the r-list for any $temp$. $\mathcal{S}$ calculates $\text{ProcOnion}(SK(P_i), O, P_i) = (O', P')$.

(a) $P' = \perp$: P_{o_j} is the recipient and O' is a message or a fail symbol. S thus sends the message $(ProcessNewOnion, P_i, O', n, ())$ to $\mathcal{F}$ on P_a's behalf and as $\mathcal{A}$ already delivered this message to the honest party sends (Deliver Message, $temp$) for the belonging $temp$ (Notice that $\mathcal{S}$ knows which $temp$ belongs to this communication as it is started at an adversarial party P_a).

(b) $P' \neq \perp$: S picks a message $m \in \mathcal{M}$. S sends on P_a's behalf the message, $Process\_New\_Onion(P', m, n, ())$ from P_i and $Deliver\_Message(temp)$ for the belonging $temp$ (Notice that $\mathcal{S}$ knows the $temp$ as in case (a)) to $\mathcal{F}$. $\mathcal{S}$ adds the entry $(O', P', (P_a, sid, P', m, n, ()))$ to the O-list.

[14] $\mathcal{S}$ knows whether they belong to an adversarial sender from the output it gets

2) Indistinguishability: **Hybrid** $\mathcal{H}_0$. This machine sets up the keys for the honest parties (so it has their secret keys). Then it interacts with the environment and $\mathcal{A}$ on behalf of the honest parties. It invokes the real protocol for the honest parties in interacting with $\mathcal{A}$.

Hybrid $\mathcal{H}_1^1$. In this hybrid, for one communication the onion layers from its honest sender to the next honest node (relay or receiver) are replaced with random onion layers embedding the same path. More precisely, this machine acts like $\mathcal{H}_0$ except that the consecutive onion layers $O_1, O_2, \ldots, O_j$ from an honest sender P_0 to the next honest node P_j are replaced with $\bar{O}_1, \ldots, \bar{O}_j$ where $\bar{O}_i = O'_{k+i}$ with $(O'_1, \ldots, O'_n) \leftarrow \mathrm{FormOnion}(m_{rdm}, \mathcal{P}_rdm, (PK)_{\mathcal{P}_{rdm}})$ where m_{rdm} is a random message, $\mathcal{P}$ a random path that includes the sequence from P_0 to P_j starting at the k-th node. $\mathcal{H}_1^1$ keeps a $\bar{O}$-list and stores $(\bar{O}_j, P_j, \mathrm{ProcOnion}(SK_{P_j}, O_j, P_j))$ on it. If an onion $\tilde{O}$ is sent to P_j, the machine tests if processing results in a fail (replay detected and dropped). If it does not, $\mathcal{H}_1^1$ compares $\tilde{O}$ to all $\bar{O}_j$ on its $\bar{O}$-list where the second entry is P_j. If it finds a match, the belonging $\mathrm{ProcOnion}(SK_{P_j}, O_j, P_j)$ is used as processing result of P_j. Otherwise, $\mathrm{ProcOnion}(SK_{P_j}, \tilde{O}, P_j)$ is used.

$\mathcal{H}_\mathbf{0} \approx_\mathbf{I} \mathcal{H}_\mathbf{1}^\mathbf{1}$. The environment gets notified when an honest party receives an onion layer and inputs when this party is done. As we just exchange onion layers by others, the behavior to the environment is indistinguishable for both machines.

$\mathcal{A}$ observes the onion layers after P_0 and if it sends an onion to P_j the result of the processing after the honest node. Depending on the behavior of $\mathcal{A}$ three cases occur: $\mathcal{A}$ drops the onion belonging to this communication before P_j, $\mathcal{A}$ behaves protocol-conform and sends the expected onion to P_j or $\mathcal{A}$ modifies the expected onion before sending it to P_j. Notice that dropping the onion leaves the adversary with less output. Thus, we can focus on the other cases.

We assume there exists a distinguisher $\mathcal{D}$ between $\mathcal{H}_0$ and $\mathcal{H}_1^1$ and construct a successful attack on LU:

The attack receives key and name of the honest relay and uses the input of the replaced communication as choice for the challenge, where it replaces the name of the first honest relay with the one that it got from the challenger[15]. For the other relays the attack decides on the keys as $\mathcal{A}$ (for corrupted) and the protocol (for honest) does. It receives $(\tilde{O}, \mathrm{ProcOnion}(O_j))$ from the challenger. The attack uses $\mathcal{D}$. For $\mathcal{D}$ it simulates all communications except the one chosen for the challenge, with the oracles and knowledge of the protocol and keys. (This includes that for bit-identical onions for which the oracle cannot be used, depending on whether the protocol has replay protection $\mathrm{ProcOnion}(O_j)$ is reused or the onion is dropped.) For simulating the challenge communication the attack hands $\tilde{O}$ to $\mathcal{A}$ as soon as $\mathcal{D}$ instructs to do so. To simulate further for $\mathcal{D}$ it uses $\tilde{O}$ to calculate the later layers and does any actions $\mathcal{A}$ does on the onion.

$\mathcal{A}$ either sends the honest processing of $\tilde{O}$ to the challenge router or $\mathcal{A}$ modifies it to $f(\tilde{O})$. In the first case, the attack simulates corresponding to $\mathrm{ProcOnion}(O_j)$. In the second case, $f(\tilde{O})$ is given to the oracle and the simulation is done for the returned $\mathrm{ProcOnion}(f(\tilde{O}))$.

Thus, either the challenger chose $b = 0$ and the attack behaves like $\mathcal{H}_0$ under $\mathcal{D}$; or the challenger chose $b = 1$ and the attack behaves like $\mathcal{H}_1^1$ under $\mathcal{D}$. The attack outputs the same bit as $\mathcal{D}$ does for its simulation to win with the same advantage as $\mathcal{D}$ can distinguish the hybrids.

Hybrid $\mathcal{H}_1^*$. In this hybrid, for one communication, for which they had not been replaced, onion layers from an honest sender to the next honest node are replaced with a random onion sharing this path.

$\mathcal{H}_\mathbf{1}^\mathbf{1} \approx_\mathbf{I} \mathcal{H}_\mathbf{1}^*$. Analogous above. Apply argumentation of indistinguishability ($\mathcal{H}_0 \approx_I \mathcal{H}_1^1$) for every replaced subpath.[16]

Hybrid $\mathcal{H}_2^1$. In this hybrid, for one communication (and all its replays) for which in the adversarial processing no modification occurred[17] onion layers between two consecutive honest relays (the second might be the receiver) are replaced with random onion layers embedding the same path. More precisely, this machine acts like $\mathcal{H}_1^*$ except that the processing of O_j (and, if no replay protection, the processing result of all replays of O_j); i.e. the consecutive onion layers $O_{j+1}, \ldots, O_{j'}$ from a communication of an honest sender, starting at the next honest node P_j to the next following honest node $P_{j'}$, are replaced with $\bar{O}_{j+1}, \ldots, \bar{O}_{j'}$. Thereby, $\bar{O}_{j+1} = O'_{j+k+1}$ with $(O'_1, \ldots, O'_n) \leftarrow \mathrm{FormOnion}(m_{rdm}, \mathcal{P}_{rdm}, (PK)_{\mathcal{P}_{rdm}})$ where m_{rdm} is a random message, $\mathcal{P}$ a random path that includes the sequence from P_j to $P_{j'}$ starting at the k-th node. $\mathcal{H}_2^1$ stores $(\bar{O}_{j'}, P_{j'}, \mathrm{ProcOnion}(SK_{P_{j'}}, O_{j'}, P_{j'}))$ on the $\bar{O}$-list. Like in $\mathcal{H}_1^*$ if an onion $\tilde{O}$ is sent to $P_{j'}$, processing is first checked for a fail. If it does not fail , $\mathcal{H}_2^1$ compares $\tilde{O}$ to all $\bar{O}_{j'}$ on its $\bar{O}$-list where the second entry is $P_{j'}$. If it finds a match, the belonging $\mathrm{ProcOnion}(SK_{P_{j'}}, O_{j'}, P_{j'})$ is used as processing result of $P_{j'}$. Otherwise, $\mathrm{ProcOnion}(SK_{P_{j'}}, \tilde{O}, P_{j'})$ is used.

$\mathcal{H}_\mathbf{1}^* \approx_\mathbf{I} \mathcal{H}_\mathbf{2}^\mathbf{1}$. $\mathcal{H}_2^1$ replaces for one communication (and all its replays), the first subpath between two consecutive honest nodes after an honest sender. The output to $\mathcal{A}$ includes the earlier (by $\mathcal{H}_1^*$) replaced onion layers $\bar{O}_{earlier}$ before the first honest relay (these layers are identical in $\mathcal{H}_1^*$ and $\mathcal{H}_2^1$) that take the original subpath but are otherwise chosen randomly; the original onion layers after the first honest relay for all communications not considered by $\mathcal{H}_2^1$ (outputted by $\mathcal{H}_1^*$) or in case of the communication considered by $\mathcal{H}_2^1$, the newly drawn random replacement (generated by $\mathcal{H}_2^1$); and the processing after $P_{j'}$.

The onions $\bar{O}_{earlier}$ are chosen independently at random by $\mathcal{H}_1^*$ such that they embed the original path between an honest sender and the first honest relay, but contain a random

[15]As both honest nodes are randomly drawn this does not change the success

[16]Technically, we need the onion layers as used in $\mathcal{H}_1^1$ (with replaced onion layers between a honest sender and first honest node) in this case. Hence, slightly different than before the attack needs to simulate the other communications not only by the oracle use and processing, but also by replacing some onion layers (between the honest sender and first honest node) with randomly drawn ones as $\mathcal{H}_1^1$ does.

[17]We treat modifying adversaries later in a generic way.

message and random valid path before the honest sending relay and after the next following honest relay. As they are replaced by the original onion layers after P_j (there was no modification for this communication) and include a random path and message, onions $\bar{O}_{earlier}$ cannot be linked to onions output by P_j. Hence, the random onions before the first honest node do not help distinguishing the machines.

Thus, all that is left to distinguish the machines, is the original/replaced onion layer after the first honest node and the processing afterwards. This is the same output as in $\mathcal{H}_0 \approx_I \mathcal{H}_1^1$. Hence, if there exists a distinguisher between $\mathcal{H}_1^*$ and $\mathcal{H}_2^1$ there exists an attack on LU.

Hybrid $\mathcal{H}_2^*$. In this hybrid, for all communications, one communication (and all its replays) at a time is selected. Within that communication, the next (from sender to receiver) non-replaced subpath between two consecutive honest nodes is chosen. If $\mathcal{A}$ previously (i.e. in onion layers up to the honest node starting the selected subpath) modified an onion layer in this communication, the communication is skipped. Otherwise, the onion layers between those honest nodes are replaced with a random onion sharing the path.

$\mathcal{H}_\mathbf{2}^\mathbf{1} \approx_\mathbf{I} \mathcal{H}_\mathbf{2}^*$. Analogous above.

Hybrid $\mathcal{H}_3^1$. In this hybrid, for one communication (and all its replays) for which in the adversarial processing no modification occurred so far, onion layers from its last honest relay to the corrupted receiver are replaced with random onions sharing this path and message. More precisely, this machine acts like $\mathcal{H}_2^*$ except that the processing of O_j (and, if no replay protection, the processing result of all replays of O_j); i.e. the consecutive onion layers $O_{j+1}, \ldots, O_n$ from a communication of an honest sender, starting at the last honest node P_j to the corrupted receiver P_n are replaced with $\bar{O}_{j+1}, \ldots, \bar{O}_n$. Thereby $\bar{O}_i = O'_{k+i}$ with $(O'_1, \ldots, O'_{n'}) \leftarrow \mathrm{FormOnion}(m, \mathcal{P}_{rdm}, (PK)_{\mathcal{P}_{rdm}})$ where m is the message of this communication[18], $\mathcal{P}_{rdm}$ a random path that includes the sequence from P_j to P_n starting at the k-th node.

$\mathcal{H}_\mathbf{2}^* \approx_\mathbf{I} \mathcal{H}_\mathbf{3}^\mathbf{1}$. Similar to $\mathcal{H}_1^* \approx_I \mathcal{H}_2^1$ the onion layers before P_j are independent and hence do not help distinguishing. The remaining outputs suffice to construct an attack on TI similar to the one on LU in $\mathcal{H}_1^*$ and $\mathcal{H}_2^1$.

Hybrid $\mathcal{H}_3^*$. In this hybrid, for one communication (and all its replays) for which in the adversarial processing no modification occurred so far and for which the onion layers from its last honest relay to corrupted receiver have not been replaced before, the onion layers between those nodes are replaced with random onion layers sharing the path and message.

$\mathcal{H}_\mathbf{3}^\mathbf{1} \approx_\mathbf{I} \mathcal{H}_\mathbf{3}^*$. Analogous above.

Hybrid $\mathcal{H}_4$ This machine acts the way that $\mathcal{S}$ acts in combination with $\mathcal{F}$. Note that $\mathcal{H}_3^*$ only behaves differently from $\mathcal{S}$ in (a) routing onions through the honest parties and (b) where it gets its information needed for choosing the replacement onion layers: (a) $\mathcal{H}_3^*$ actually routes them through the real honest parties that do all the computation. $\mathcal{H}_4$, instead runs the way that $\mathcal{F}$ and $\mathcal{S}$ operate: there are no real honest parties, and the ideal honest parties do not do any crypto work. (b) $\mathcal{H}_3^*$ gets inputs directly from the environment and gives output to it. In $\mathcal{H}_4$ the environment instead gives inputs to $\mathcal{F}$ and $\mathcal{S}$ gets the needed information (i.e. parts of path and the included message, if the receiver is corrupted) from outputs of $\mathcal{F}$ as the ideal world adversary. $\mathcal{F}$ gives the outputs to the environment as needed. Further, $\mathcal{H}_3^*$ chooses the replacement onion layers randomly, but identical for replays, while $\mathcal{S}$ chooses them pseudo-randomly depending on an in $\mathcal{F}$ randomly chosen $temp$, which is identical for replays.

$\mathcal{H}_\mathbf{3}^* \approx_\mathbf{I} \mathcal{H}_\mathbf{4}$. For the interaction with the environment from the protocol/ideal functionality, it is easy to see that the simulator directly gets the information it needs from the outputs of the ideal functionality to the adversary: whenever an honest node is done processing, it needs the path from it to the next honest node or path from it to the corrupted receiver and in this case also the message. This information is given to $\mathcal{S}$ by $\mathcal{F}$.

Further, in the real protocol, the environment is notified by honest nodes when they receive an onion together with some random ID that the environment sends back to signal that the honest node is done processing the onion. The same is done in the ideal functionality. Notice that the simulator ensures that every communication is simulated in $\mathcal{F}$ such that those notifications arrive at the environment without any difference.

For the interaction with the real world adversary, we distinguish the outputs in communications from honest and corrupted senders. 0) Corrupted senders: In the case of a corrupted sender both $\mathcal{H}_3^*$ and $\mathcal{H}_4$ (i.e. $\mathcal{S}$+$\mathcal{F}$) do not replace any onion layers except that with negligible probability a collision on the $\bar{O}$-list resp. O-list occurs.

1)Honest senders: 1.1) No modification of the onion by the adversary happens: All parts of the path are replaced with randomly drawn onion layers $\bar{O}_i$. The way those layers are chosen is identical for $\mathcal{H}_3^*$ and $\mathcal{H}_4$ (i.e. $\mathcal{S}$+ $\mathcal{F}$). 1.2) Some modification of the onion or a drop or insert happens: As soon as another onion as the expected honest processing is found, both $\mathcal{H}_3^*$ and $\mathcal{H}_4$ continue to use the bit-identical onion for the further processing except that with negligible probability a collision on the $\bar{O}$-list resp. O-list occurs. In case of a dropped onion it is simply not processed further in any of the two machines.

Note that the view of the environment in the real protocol is the same as its view in interacting with $\mathcal{H}_0$. Similarly, its view in the ideal protocol with the simulator is the same as its view in interacting with $\mathcal{H}_4$. As we have shown indistinguishability in every step, we have indistinguishability in their views.

C. Sphinx

1) Adapted Sphinx: The original Sphinx protocol was adapted in [5] to use modern cryptographic primitives, which can be proven secure. Further, the number of different cryptographic algorithms is reduced to improve performance of the construction. Additionally, the encryption function used for the Sphinx payload is replaced by an authenticated encryption (AE) scheme, such that the payload is also authenticated at

[18] $\mathcal{H}_3^1$ knows this message as it communicates with the environment.

each node by the tag γ_i as part of the header. Let π_{AE} (π_{AE}^{-1}) be the encryption (decryption) function of an AE scheme, as proposed by [5].

The algorithm to generate a Sphinx packet is partly adapted. Calculation of $\alpha_i, s_i, b_i, \beta_i$ is equivalent to the original Sphinx description, except that we consider the 0-bit string for padding $\beta_{\nu-1}$ replaced by random bits to prevent the known attack from Section IV-A. The cryptographic primitives μ, h_μ, π, h_π are not used anymore in the adaptation. Instead an AE scheme is employed: Let δ_ν be the payload of the Sphinx packet. For $0 \leq i < \nu - 1$: $(\delta_i, \gamma_i) \leftarrow \pi_{AE}(s_i, \delta_{i+1}, \beta_i)$, where δ_i is an encryption of δ_{i+1} and γ is a tag authenticating δ_{i+1}, β_i. $\pi_{AE}, \rho, h_b, h_\rho$ are modelled as a random oracle. The length of the Sphinx payload is fixed and checked at all mix nodes. If the length is incorrect, the packet is discarded.

2) Proof of adapted Sphinx: The proof for Onion-Correctness is analogous to the one in [13]. The proof of our new security properties follows:

Symmetric key s_i is a secret: The mix nodes have an asymmetric private key x_{n_i}, that is used in a Diffie-Hellman key exchange. It follows that the shared symmetric key between an honest sender and an honest mix node is not known to the adversary. If an adversary could extract the symmetric key with non-negligible probability, she could break the decisional diffie-hellman problem. See [13] Section 4.4, indistinguishability proof of hybrid $\mathbf{G}_1$. Note that tag γ is generated using an AE scheme keyed with s_i directly. The argumentation from [13] still holds.

LU : Recall that LU allows the adversary to decide the inputs to $\mathrm{FormOnion}$ and either returns the resulting onion O_1 of this $\mathrm{FormOnion}$ call or a randomly chosen onion $\bar{O}_k$, that only matches the subpath between the honest nodes, together with the processing of O_1 after the honest node ($\mathrm{ProcOnion}(O_j)$). Furthermore, it allows oracle use before and after this decision.

No dependencies between $\mathrm{FormOnion}$*:* We define the game LU^1 to be the same as LU except that the adversary has no oracle access before his input decision (skips Step 2). As the creation of onions in Sphinx is adequately randomized, independent from earlier creations and using a sufficiently large security parameter, oracle access before the challenge only negligibly improves the adversary's success in guessing correctly.

No modification: We define the game LU^2 to be the same as LU^1 except that the adversary has no oracle access after his input decision (skips Step 7). Using the oracle for a new onion $\tilde{O}$ independent of the challenge onion O does not help guessing b as the output $\mathrm{ProcOnion}(\tilde{O})$ is then independent from b as well. Thus, we only need to look at modifications of the challenge onion processed until the honest node $O_{+j} := \mathrm{ProcOnion}^j(O)$. As any onion layer, O_{+j} consists of four parts $(\alpha, \beta, \gamma, \delta)$, from which the tag γ authenticates β, δ using a shared key s extracted from α. Modifications generating a valid tag are thus only successful with negligible probability. Therefore, there cannot be a successful attack on LU^1 that relies on the second oracle and thus any successful attack on LU^1 is also possible for LU^2 in Sphinx.

No linking: We define the game LU^3 to be LU^2 but the second part of the output ($ProcOnion(O_j) = (O_{j+1}, P_{j+1})$) is no longer given to the game adversary. Assume knowing this output helps the adversary to break LU. As the next hop P_{j+1} is already known to her from her choice of path, the only part of the output that can help her is O_{j+1}. Thus the adversary must be able to link O_{j+1} to the first output onion layer (O_1 resp. $\bar{O}_k$) which differs depending on b.

Hence, she must be able to link the onion layers before and after the honest node. The processing at a honest node changes all four parts of a Sphinx packet in a way such that the adversary cannot predict the result. Let $B = (\beta \| 0_{2\kappa}) \oplus \rho(h_\rho(s))$: $\alpha' \leftarrow \alpha^{h_b \alpha, s}$; $\beta' \leftarrow B_{[2\kappa..(2r+3)\kappa-1]}$; $\gamma' \leftarrow B_{[\kappa..2\kappa-1]}$; $\delta' \leftarrow \pi_{AE}^{-1}(s, \delta, \gamma)$. Assume if the adversary can decide on $(\alpha, \beta, \gamma, \delta)$ she can distinguish any of the new values $(\alpha', \beta', \gamma', \delta')$ from randomness without knowing s. However, this implies that she is able to solve the DDH problem induced by the computation for α', or break the secure ρ, π_{AE}, or hash primitives, which contradicts the assumption. Thus, no successful attack on LU^2 based on the second part of the output ($ProcOnion(O_j)$) can exist for Sphinx.

Onion layer indistinguishable from random ones: We define LU^4 to be LU^3 except that for the output onion layer the values of α, β, γ and δ are chosen randomly from their corresponding spaces, such that they result in the same subpath as given by the adversary. We show that LU^4 is indistinguishable from LU^3. Assume an adversary that can distinguish the games. As processing of onion layers results in expected behavior, she must be able to distinguish some of the parts of the onion layer from randomness. Assume she can distinguish any part of the packet, that means she can – without knowing s – either solve the DDH problem or break the security of ρ or the AE scheme. Therefore, she cannot distinguish any part of the packet from a randomly drawn value, and also not process it to get the message.

In LU^4 all the values are drawn exactly the same way independent of b. There cannot be an adversary with any advantage for this game. Because $LU^4 \approx LU^3 \implies LU^2 \implies LU^1 \implies LU$, we have proven that any adversary has at most negligible advantage in guessing b for LU.

TI : Recall that TI either outputs the processing of the onion build from the adversary's choice ($\mathrm{ProcOnion}(O_j) = (O_{j+1}, P_{j+1})$) or the processing from a random onion that matches the end of the path and message of the adversary's choice ($\mathrm{ProcOnion}(\bar{O}_k) = (\bar{O}_{k+1}, P_{j+1})$). Note that the next hop is always the same in those outputs and thus only the onion layers need to be indistinguishable. The proof of this is similar to LU's "Onion layer indistinguishable from random ones" except that O is chosen randomly from the onion layers that also include the adversary chosen message. Further, thanks to the fix to the attack determining the path length, also the values $\alpha_{\nu-1}, \beta_{\nu-1}, \gamma_{\nu-1}, \delta_{\nu-1}$ the last node gets are indistinguishable from such random ones.