Formal Foundations for Anonymous Communication

Zur Erlangung des akademischen Grades einer

Doktorin der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Christiane Kuhn

Tag der mündlichen Prüfung: 10. März 2022

1. Referent: Prof. Dr. Thorsten Strufe
2. Referentin: Prof. Dr. Carmela Troncoso

Abstract

With every online action, we leave our digital footprints. These massive amounts of data are used by companies and governments to derive private information and to manipulate us for their benefit. As a countermeasure to this development, anonymous communication networks have been proposed to enhance our privacy online. However, their common formal foundations are insufficient and therefore comparisons between different approaches are very limited.
With a common ground for all researchers and designers of anonymous communication networks, misunderstandings can be prevented and thus the development of networks can be enhanced. With a way to compare solutions, the optimal approach for each use case can be identified and hence development efforts no longer need to be spread across multiple projects rashly. With extensive formal foundations, we gain all of the above. Even further, formal foundations enable a deeper understanding of the possibilities, the impossibilities and the effects of the technologies used in anonymous communication networks.

In this thesis, we first create general, fundamental formalizations for anonymous communication, before we focus on onion routing and mix networks – the practically most common technique for anonymous communication.
We ensure comparability between the offered privacy, by formally defining and comparing the privacy goals. Thereby, we create the first extensive hierarchy of unambiguous privacy notions for anonymous communication. By analyzing the existing networks, we then identify basic building blocks and investigate their protection as the effect in our hierarchy of notions.
These foundations allow us to uncover and resolve existing conflicts and flaws in the related work. For privacy goal definitions, we show that based on the same informal definition two works state formal versions with considerably different protection. Moreover, we make use of the notions to compare existing impossibility results for anonymous communication: we provide the first complete picture of known limitations and bounds in anonymous communication. With the help of our in-depth approach, we are even able to tighten some of the bounds, as well as to relate them to practical state-of-the-art protocols. Finally, due to our general investigation of proposed networks and their building blocks, we discover a new attack on the predominant class of anonymous communication networks: onion routing and mix networks.
Motivated by the discovered attack, the second part of this thesis considers onion routing and mix networks. Precisely, we examine their formal foundations and practically proposed solutions to discover that our attack is partially due to a mistake in a common proof strategy for such networks. To prevent attacks, we propose a new, secure proof strategy. Further, we are able to apply our strategy for a proposed, but rarely used packet format and thus prove it secure. This packet format does however not support replies, which is likely one of the reasons why state-of-the-art solutions rely on a vulnerable packet format instead. We design two conceptual solutions for repliable, provably secure onion routing packet formats towards closing this gap.

As additional contributions, we partially relate our general results for anonymous communication to similar, but so far mostly distinct research areas, namely to privacy on the physical layer, proximity tracing applications and privacy-preserving payment systems.

Zusammenfassung

Mit jeder Online-Tätigkeit hinterlassen wir digitale Fußspuren. Unternehmen und Regierungen nutzen die privaten Informationen, die von den riesigen Datenmengen der Online-Spuren abgeleitet werden können, um ihre Nutzer und Büger zu manipulieren. Als Gegenmaßnahme wurden anonyme Kommunikationsnetze vorgeschlagen. Diesen fehlen jedoch umfassende formale Grundlagen und folglich ist der Vergleich zwischen verschiedenen Ansätzen nur sehr eingeschränkt möglich.
Mit einer gemeinsamen Grundlage zwischen allen Forschern und Entwicklern von anonymen Kommunikationsnetzen können Missverständnisse vermieden werden und die dringend benötigte Entwicklung von den Netzen wird beschleunigt. Mit Vergleichbarkeit zwischen den Lösungen, können die für den jeweiligen Anwendungsfall optimalen Netze besser identifiziert und damit die Entwicklungsanstrengungen gezielter auf Projekte verteilt werden. Weiterhin ermöglichen formale Grundlagen und Vergleichbarkeit ein tieferes Verständnis für die Grenzen und Effekte der eingesetzten Techniken zu erlangen.

Diese Arbeit liefert zuerst neue Erkenntnisse zu generellen Formalisierungen für anonyme Kommunikation, bevor sie sich dann auf die praktisch am meisten verbreitete Technik konzentriert: Onion Routing und Mix Netzwerke.
Als erstes wird die Vergleichbarkeit zwischen Privatsphärezielen sichergestellt, indem sie formal definiert und miteinander verglichen werden. Dabei enteht eine umfangreiche Hierarchie von eindeutigen Privatsphärezielen. Als zweites werden vorgeschlagene Netzwerke analysiert, um deren Grundbausteine zu identifizieren und deren Schutz als Auswirkung in der Hierarchy zu untersuchen.
Diese Grunlagen erlauben Konflikte und Schwachstellen in existierenden Arbeiten zu entdecken und aufzuklären. Genauer zeigt sich damit, dass basierend of derselben informalen Definition verschieden stark schützende formale Versionen entstanden sind. Weiterhin werden in dieser Arbeit die Notions genutzt um existierende Unmöglichkeitsresultate für anonyme Kommunikation zu vergleichen. Dabei wird nicht nur die erste vollständige Sicht auf alle bekannten Schranken für anonyme Kommunikationsnetze gegeben, sondern mit einem tiefgründigen Ansatz werden die existierenden Schranken auch gestärkt und zu praktischen, dem Stand der Kunst entsprechenden Netzen in Bezug gesetzt. Letztlich konnten durch die generellen Betrachtungen von vorgeschlagenen Netzwerken und ihren Grundbausteinen, insbesondere auch Angriffe auf die vorherrschende Klasse von anonymen Kommunikationsnetzen gefunden werden: auf Onion Routing und Mix-Netzwerke.
Davon motiviert wurden als zweiter Teil dieser Arbeit die formalen Grundlagen und praktisch eingesetzten Lösungen for Onion Routing und Mix-Netzwerke untersucht. Dabei wurde festgestellt, dass die bereits erwähnten Angriffe teilweise auf eine fehlerhafte, aber weit verbreitete Beweisstrategie für solche Netze zurückzuführen sind und es wurde eine sichere Beweisstrategie als deren Ersatz vorgeschlagen. Weiterhin wurde die neue Strategie für ein vorgeschlagenes, aber bisher nicht weiter verwendetes Paketformat eingesetzt und dieses als sicher bewiesen. Dieses Paketformat unterstützt allerdings keine Rückantworten, was höchstwahrscheinlich der Grund ist, aus dem sich aktuelle Netze auf ein unsicheres Paketformat verlassen. Deshalb wurde im Rahmen dieser Arbeit eine konzeptuelle, sichere Lösung für Onion Routing mit Rückantworten entworfen.

Als weitere verwandte Beiträge, zeigt die Arbeit Beziehungen von Teilen der generellen Ergebnisse für anonyme Kommunikationsnetze zu ähnlichen, aber bisher hauptsächlich getrennt betrachteten Forschungsbereichen, wie Privatsphäre auf der Bitübertragungsschicht, Kontaktnachverfolgung und privatsphäre-schützenden, digitalen Bezahlsystemen.

Acknowledgements

As any important project in life, this thesis was only possible due to the support of a series of people. I am extremely grateful to any single one of them for accompanying me on (a part of) my PhD journey.

First of all, I want to thank my adviser Thorsten Strufe for pointing me to anonymous communication and at the same time allowing me the freedom to shape this topic to my liking. I am grateful for his incredible patience and strong will to understand the theoretical and formal aspects that fascinate me. Of course I also thank him for the countless meetings with valuable advice, feedback and guidance and for making this thesis and all collaborations possible.

Secondly, I am extremely thankful to Carmela Troncoso not only for accepting to be the second referee for this thesis, but also for her encouraging and cordial words at my first time at PETS and, of course, for her inspiring work.

Next, I thank all of my collaborators for their contributions to the presented results and the pleasant, insightful discussions we had over the time. In particular, I thank Martin Beck for the long-continued cooperation and all the rewarding time that I got to enjoy with a mind that thinks so much alike, yet has its very own perspective. I thank Aniket Kate for allowing me to stay with him at Purdue University over a summer, for the insightful discussions to privacy-preserving payment protocols and also for valuable non-technical advice. Further, I thank Dennis Hofheinz and Andy Rupp for their great ideas on cryptographical primitives for repliable onion routing and their continuing support. Further, I also thank Stefan Schiffner for his guidance and remarks to the our privacy notions. Additionally, I want to thank Eduard Jorswieck and Pin-Hsun Lin for their patience in explaining the physical layer privacy definitions, as well as in understanding the peculiarities of the anonymous communication definitions. Further, I thank Lisza Zeidler for the long time collaboration on anonymous communication primitives and Friederike Kitzing for her contributions to the comparison of the performance bounds.

Furthermore, I thank my colleagues for their helpful comments to my texts and presentations. Especially, I am grateful for Martin Byrenheid's, Stefan Köpsell's and Clemens Deußner's inputs that made my results more accessible.

Finally, I want to thank my family and friends for their understanding and loving support during the better and especially also more challenging parts of my PhD process.

Thank you all! Sharing this experience with you really made a difference.

Contents

List of Figures

1. Introduction

Whenever we communicate online, we leave digital footprints. From these traces often private information can be derived. We reveal our interests as we like and follow in online social networks, we show our recent breakup by a drastic change in our usage behavior of our favorite messaging application and we seek medical advice and counseling on search engines. These examples are, however, just the tip of the iceberg. Due to the collection of massive amounts of data and the usage of modern machine learning algorithms, we as human beings become almost completely transparent [146] for big companies and governments, leaving us vulnerable for manipulation according to their goals. Targeted advertisement of dangerous goods, like diet products for anorexic people, for the gain of profits and targeted advertisement of political parties for the gain of power [30] are logical consequences. Furthermore, the extensive surveillance threatens our society as a whole. Nationwide censorship of critics can be used to prearrange the public opinion and even the pure awareness of the surveillance possibilities might cause minorities, journalists and whistleblowers to self-censor for their own protection [144].

To prevent the extensive surveillance power that is in the hand of a few big players, we need to protect the data and metadata of our communications. An important tool for our protection are anonymous communication networks (ACNs). However, to enhance our privacy, these networks decrease the performance as compared to communication without protection. Any ACN consequently offers some trade-off between the achieved protection and the necessary overhead. The achieved protection is thereby a combination of the privacy goal, i.e. which information the ACN keeps private, and the adversary model, i.e. against whom the privacy is guaranteed. A typical setting is to hide who sent a message to whom against the global passive adversary that can eavesdrop on all links. A variety of ACNs, from which each offers individual trade-offs in the above dimensions, has been proposed [43, 44, 58, 140, 74, 42, 40, 47, 104, 137, 154, 103, 156, 49]. Typically they are distinguished in two classes of protocols: onion routing- and mixing-based approaches [43, 44, 58, 74, 42, 140, 104, 103, 154] as opposed to secret sharing strategies [40, 47, 137, 156, 49]. Onion routing and mix networks employ intermediate hops between the communication partners to unlink the sender from the message, as well as the receiver. Secret sharing strategies instead make use of mathematical properties to ensure that any user needs to contribute a part to reveal the message.

While the development of concrete solutions for specific use cases has value in itself, the existing limits in the comparability of proposed approaches hamper the alignment of development efforts. Moreover, without an accepted, formal common ground misunderstandings between the researchers as well as developers cannot be prevented. We thus need to strengthen the existing foundations via a detailed investigation of all dimensions and techniques to further the development of ACNs. Practically, the gained fundamental understanding is not only important to exclude wrong directions with impossibility results and create completely new solutions following more promising approaches, but also to prove and improve the protection of existing networks. Indeed, investigating the achieved trade-off in detail allows to discover attacks and flaws in the designs.

For this thesis, we first show general results for anonymous communication, before we then investigate onion routing and mix networks as concrete practical examples. To improve the general, theoretical foundations for ACNs, we focus on i) privacy goals and ii) performance limits.

Firstly, the published ACNs address a variety of *privacy goals*. However, especially the goal definition is usually ad hoc or informal. This leads to a situation where authors disagree on the interpretation of informal definitions. For example "Sender Unlinkablity" of Hevia and Micciancio's framework [82] and "Sender Anonymity" of AnoA [11] are both claimed to be equivalent to "Sender Anonymity" of Pfitzmann and Hansen's terminology [126], but significantly differ in the protection they actually provide. To rigorously compare the ACNs, their goals need to be unambiguously defined and the relations between the goals understood. As our first contribution, we formalize the privacy goals for anonymous communication. We build our privacy definitions on the foundations of existing analytical frameworks [11, 26, 82, 72] that rely on indistinguishability games. Indistinguishability games are known from message confidentiality [17]. There an adversary, seeing only the encrypted message, should not be able to exclude any of the two chosen plaintext message as a candidate. Similarly, we require that given her observations, the adversary cannot conclude any private information about the communication, e.g. which of the two chosen users the sender was. This approach allows us to vary the kind of information that is supposed to be private for any specific notion. We define an extensive set of communication properties that are (depending on the notion) either declared private, i.e. must be hidden from the adversary, or not. With the help of these formal definitions, we compare all identified privacy notions, to learn which is stronger than the other and present a hierarchy of them. Further we relate our notions to formal existing work [11, 26, 82, 72] and thereby solve naming conflicts, like the different interpretations of "Sender Anonymity" that were mentioned above. Moreover, we identify and investigate basic building blocks that ACNs are often constructed of. Our goal is to learn about their inner workings and the protection achieved by the building blocks in terms of our notions. So far, to the best of our knowledge only a rough classification of approaches is common and only Hevia and Miccianchio [82] proposed three strategies that represent basic building block style techniques. While we of course base our investigation on this seminal work, we provide a notably more fine-granular set of building blocks against the global passive adversary.

Secondly, the published ACNs are bound by *performance restrictions*. Especially, some goals cannot be achieved unless a certain performance overhead is invested. The knowledge of such fundamental limits can greatly aid the ACN design. However, the existing valuable formal results [6, 54, 72, 82] are incomparable and hard to understand in detail because of the complex dimensions of ACNs. With the help of our privacy goals hierarchy and in-depth understanding of the impossibility results for ACNs, we compare all existing performance bounds [6, 54, 72, 82] in detail. For our comparison, we unify the differing models and privacy goals, as well as handle the large variety of explicit and implicit assumptions. Furthermore, we strengthen some of the bounds' claims and proofs during our investigation. We contextualize our final findings by relating them to state-of-the-art ACN proposals and we identify theoretical peculiarities that cause high overhead, but give grounds for more realistic and relaxed bounds in future work.

During our search for building blocks, we broadly investigated state-of-the-art ACNs regarding their protection in terms of our privacy notions. Thereby, we discovered the *malleability attack* on HORNET [43], an onion routing based solutions. HORNET aims to unlink senders from receivers with the help of a series of intermediate nodes. The payload in HORNET is however merely end-to-end, not hop-by-hop integrity protected in the used packet format Sphinx [52]. Thus a packet with modified payload still reaches the intended receiver, which is exploited in our malleability attack. More precisely, the attack assumes a corrupt first intermediate node and receiver. The adversary links the sender to the receiver by modifying the payload of the sender victim and recognizing the uncommon message (a random bit string) at the adversarial receiver.

Motivated by the malleability attack, we focus on onion routing and mix networks for the second part of this thesis. The most prominent representative of onion routing, Tor [58], gained users over time and now has an impressive user base of around two million users[1]. As onion routing and mix network solutions became increasingly popular, the need to formally prove their privacy became apparent. Besides others, Camenisch and Lysyanskaya [31] proposed a fundamental security definition for onion routing, in the form of an ideal functionality. This functionality is a formal abstraction that shows which information even a perfect onion routing scheme reveals to an adversary. Additionally the authors design easier to prove game-based properties which, as they claim, provide the same protection as the ideal functionality. The properties of Camenisch and Lysyanskaya have been used by many approaches, especially by the fundamental packet format

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

Sphinx.

Investigating the security of the proposed onion routing and mix networks, we discovered that the malleability attack works on nearly[2] all networks [43, 52, 142, 128] that are proposed and proven secure with the properties of Camenisch and Lysyanskaya. For HORNET it even allows the adversary to recover parts of the message additionally to the sender-receiver relationship. The attack is surprising, as the networks were proven secure. We hence investigated the proofs and indeed had to discover that the proposed properties are not sufficient for the privacy promised in the ideal functionality. To close this gap, we designed new properties that indeed imply the ideal functionality. Further, we show for one packet format with explicit hop-based payload protection [14] that it fulfills our properties.

However, this packet format, as well as the model of Camenisch and Lysyanskaya does not support replies from the (potentially adversarial) receiver back to the anonymous sender. Practical solutions [43, 52, 142, 128] on the other hand typically require replies from the receiver to the original sender. We thus extend the model and proof strategy with replies. To provide sufficient protection for users that request replies, even if few messages are being replied to, it is desirable to make forward and backward communications indistinguishable. This however comes with a challenge when the malleability attack needs to be prevented: It requires payload protection not only for requests as before, bot now also for replies, which can only be realized implicitly without trivially revealing the linking between sender and receiver. Overcoming the technical challenge of *implicit* payload protection for the reply, we propose two new packet formats that conceptually solve the problem of repliable and provably secure onion routing.

Beyond our contributions to anonymous communication, we made use of the gained knowledge on privacy goal definitions in related areas. We thus proposed a notion hierarchy tailored to proximity tracing applications, as well as privacy-preserving payment systems. For both use cases, we further related proposed solutions to our new hierarchies. Additionally, we found equivalences of our notions to existing formal definitions that are used on the physical layer and reveal a first connection point between these research areas.

The presented contributions have been published at several top venues, including IEEE S&P [95], Asiacrypt [97], PETS [93], and WPES [99]. For this thesis, the corresponding publications are adapted. Below we give a concise overview of the included contributions.

Contributions

Our contributions to the formalization of anonymous communication networks can be classified into two parts: the fundamental general theoretical insights and the more practical results to onion routing and mix network packet formats.

1. *General Insights.* i) Definition and comparison of privacy goals, ii) investigation of basic protection techniques and iii) comparison of performance bounds

2. *Onion routing and mix networks.* i) Identification of attacks on and flaws in state-of-the-art networks and packet formats, ii) identification of flaws in the underlying proof techniques, iii) proposition of secure proof strategies and iv) proposition of secure packet formats

Beyond anonymous communication we investigated the relation to i) physical layer, ii) proximity tracing applications and iii) payment systems.

General Insights

Privacy Goals (Chapter 3). We present privacy notions for anonymous communication created out of basic building blocks: the properties of communications. These properties are (depending

[2]Only for TARANET [44] the attack is outside of the adversary model.

on the notion) either private and have to be protected, or not. We compare each pair of defined notions to create an extensive hierarchy and further relate existing work to it to resolve conflicts. Moreover, we identify and investigate a fine-grained set of basic techniques against the global passive adversary.

Performance Bounds (Chapter 4). We investigate the performance bounds in detail, tighten some of them and bring them to a common model for comparison. Thereby, we are able to state the complete picture of known impossibility results for ACNs. By relating them with state-of-the-art ACN proposals, we even identify limitations of the theoretical bounds and opportunities for future work.

Onion Routing and Mix Networks

Onion Routing and Mix Networks (Chapter 5). During our investigation of state-of-the-art solutions, we had to discover the malleability attack on onion routing and mix networks. We argue that the malleability attack works on a series of onion routing and mix networks, even though they have been proven secure. We hence investigate the underlying proof strategy to find the mistakes and propose a corrected proof strategy. Moreover, we apply the new proof strategy to an existing packet format.

Adding Replies (Chapter 6). The model behind the proof strategy, as well as the mentioned packet format does not support replies. We hence extend the model and proof strategy to cover replies. Further, we overcome a technical challenge that arises from the combination of reply support and our malleability attack, and design two conceptual solutions for secure repliable onion routing packet formats.

Related Contributions

Additionally to our fundamental results for ACNs, we relate these results partially to other application areas, namely proximity tracing applications and payment systems.

Proximity Tracing (Section 7.1). Due to the contemporary challenge of the covid pandemic, we decided to aid the design of privacy preserving proximity tracing applications with clear and formally defined privacy notions for this setting. We further investigate the proposed approaches regarding the offered privacy in terms of our definitions.

Privacy Preserving Payment Systems (Section 7.2). Payments are conceptually similar to communications. In both cases a sender sends something (message or money) to a receiver. We thus studied the differences between the research areas, defined privacy notions for payments and investigated state-of-the-art solutions, as well as own conceptual adaptations for their achieved privacy.

Additionally, with the help of our formal privacy notions for ACNs, we were able to show equivalences between these definitions and physical layer privacy definitions (see [106] for details).

Collaborations

In order to achieve the presented results, I was fortunate to work with many talented researchers. As I am thus presenting the outcome of our collaborations, I am using “we” not out of convention, but to honor the support and efforts of my valued collaborators.

Precisely, my supervisor Thorsten Strufe supported me with his input in countless discussions to all my research. Further, Martin Beck contributed to our privacy goal definitions, the onion routing formalization and the proximity tracing privacy requirements in numerous, long discussions. Especially, Martin Beck and I discovered the first instance of the malleability attack in a joint discussion and we fine-tuned the new onion routing security properties and the corresponding proof together. Stefan Schiffner and Eduard Jorswieck contributed to the property definitions for our notions by helping to find the most elegant representation for them. Pin-Hsun Lin and Eduard Jorswieck further contributed the majority of results for the comparison of our anonymous communication notions to the physical layer definitions. Friederike Kitzing contributed to the comparison of impossibility results. Lisza Zeidler contributed to the basic building blocks for anonymous communication. Without Andy Rupp and Dennis Hofheinz I would not have found the cryptographic primitives necessary to build the repliable onion routing and mixnet packet format, neither written the current version of the security proof for the new schemes. Aniket Kate contributed to the privacy goals for privacy-preserving payment systems and the relation of goals to state-of-the-art payment protocols.

2. Preliminaries

In this chapter, we give general background on the setting of ACNs, selected underlying cryptographic primitives, and a concise overview on the necessary privacy goal background. We will further detail the background and state related work in the corresponding chapters together with the new contributions.

2.1. Anonymous Communication Networks

While encryption is a well known measure to protect the content of messages, packet switched networks leak other information that requires protection. For example the sender and receiver of a communication are important metadata information that allows conclusions about social relationships. ACNs are tools that exchange messages between users and hide such communication data and metadata from the adversary.

2.1.1 Setting

We discuss Anonymous Communication (AC) in the setting of multiple unicast communications, like several client-server applications or messaging between users on the Internet. Each communication is the process of delivering a single message from the sender to the intended receiver. To achieve this, the message is possibly forwarded by intermediate nodes. ACNs consist of nodes playing two roles; users (senders and receivers) and service providers (intermediate nodes). Participants in some systems play both roles for different communications. We distinguish between the *integrated system model*, where the receiver is part of the ACN and acts according to the ACN protocol, and the *service model*, where messages are anonymized as a service and the receiver, e.g. webserver, can be completely unaware of the ACN. Depending on the use case, the assumed adversaries and the privacy goals differ (see below).

2.1.2 Adversary

ACNs vary widely in the adversarial capabilities they assume. Most well known is the global passive adversary (GPA) that allows to eavesdrop on all network links. More strict adversary models also consider corruption of users and intermediate nodes, as well as active attacks. While active adversaries allow to modify, drop, insert and delay packets at the controlled parts of the network, passively corrupted receivers, or intermediate nodes only leak their secrets, e.g. private keys and observations, to the adversary. Additionally, adversary models might limit the adversary to local reach, i.e. to eavesdrop or modify packets only at some part of the network, to certain time periods or to ignore time measurements.

2.1.3 Goals

ACNs tackle a wide range of privacy goals. Most commonly approaches want to protect who is sending which message ("Sender Anonymity"), who is receiving which message ("Receiver Anonymity") and who is communicating with whom ("Relationship Anonymity"). All users that the adversary cannot distinguish from the actual sender of a certain communication build the sender *anonymity set* for this communication (similar for the receiver). While these goals cover the most intuitive protection, indeed sequences of communications contain more metadata information. Fine grained privacy goals thus consider a variety of properties, like e.g. whether the sending frequency or the connection between messages of the same sender is hidden.

2.1.4 Performance

The increase of privacy provided by ACNs comes with additional overhead as compared to networks without privacy protection. While many metrics exist, for the theoretical performance limits discussed in this thesis, we follow the convention set by the related work[1] [54] and focus on two kinds of overhead: bandwidth and latency.

Bandwidth overhead accounts for the number of additionally sent packets and *latency overhead* accounts for the number of additionally taken hops.

Further, for packet formats, we discuss the overhead as the number of additionally transferred bits and processing time at intermediate nodes.

2.1.5 Techniques

We distinguish indirection, secret sharing and dummy traffic techniques to enhance communication privacy.

Indirection. Indirection-based techniques hide the connection of a sender to her message and receiver by relaying the message over multiple hops. Applying techniques like layered encryption or shuffling of messages, they ensure unlinkablity of incoming and outgoing messages at honest intermediate nodes.

Prominent examples are *onion routing* [58, 74], which uses layered encryption and a tunnel setup to ensure that the communication is not identifiable based on the message while being performant, and *mix networks* [42], which in addition to layered encryption employ random delays at each hop to hide timing correlations. *Free-route* networks allow the sender to choose the intermediate hops without any restrictions, while *cascade* and *layered* designs restrict which combinations of intermediate hops can be chosen.

Onion routing networks protect the sender-receiver and sender-message relationships against passive adversaries that corrupt all but one of the intermediate hops if timings are not considered. Otherwise, observations before and after the honest intermediate node can be linked based on their timings. Mix networks with their random delays allow the above protection even if the timings are observed (e.g. by a GPA). Further, as active attacks at adversarial nodes are likely, additional extensions, like detection of dropped packets with looping messages or detection of replayed packet with duplicate filters, are often employed to provide protection against stronger adversaries (see e.g. [6, 44, 128]).

Secret sharing. Secret sharing based techniques make use of mathematical properties to ensure that a message can only be recovered if information that potentially belongs to many different users

[1]We note that these metrics for performance limits in ACNs differ from metrics used for other communication network applications.

or communications is accessed. Prominent examples are DC-Nets [40] and Private Information Retrieval (PIR) [47].

DC-Nets [40] implement secret sharing to broadcast one message per round while ensuring that any participant is sending a part that is necessary to recover the message. Thereby they hide the sender under all honest participants against an adversary that observes or even participates in the protocol. However, sending more than one message per round leads to collisions of messages and none of them are interpretable. A collision avoidance scheme is thus often assumed.

Private information retrieval (PIR) [47] allows to request and deliver an entry of a database without disclosing to the database owner which entry was requested. Using e.g. superposed shares of data, it allows a receiver to anonymously request messages that are stored at the database. Even the adversary that owns the database cannot conclude which message the receiver retrieved or with whom the receiver is communicating.

Dummy messages. *Dummy messages* do not transmit useful information. They instead are sent to hide sending and receiving of "*real messages*", which contain useful information for a receiver. Dummy messages are sent randomly, or systematically according to user synchronization. Later in the protocol they are dropped by some entity, usually an intermediate node or the receiver.

A prominent example is *broadcasting with implicit addressing* [127]. There the encrypted message is sent to all receivers and each receiver checks if the message is intended for her based on the implicit address (e.g. by trying to decrypt the message with her private key). This hides the intended receiver from a global passive adversary because everyone receives the same encrypted message.

2.2. Cryptographic building blocks

As typical for ACNs, our constructions will rely on cryptographic building blocks. We informally introduce the building blocks here and refer to Appendix C.4.1 for formal definitions of them.

2.2.1 Encryption

For our constructions we use PRP-CCA (PseudoRandom Permutation - Chosen Ciphertext Attack) secure symmetric and CCA2 (adaptive Chosen Ciphertext Attack) secure, as well as rerandomizable CPA (Chosen-Plaintext Attack) secure asymmetric encryption. We further discuss key-private asymmetric encryption as a building block.

An encryption scheme consists of a key generation, an encryption and a decryption algorithm. Symmetric encryption schemes (also called symmetric key encryption (SKE)) use the same key for encryption and decryption. Asymmetric schemes (also called public key encryption (PKE)) on the other hand generate a key pair of a public key (used for encryption) and private key (used for decryption). For correctness, we require that the decryption of an encrypted message with the correct key always results in the used message.

For *PRP-CCA secure symmetric encryption* [16] , we restrict the class of symmetric encryption schemes to the ones where the encryption function for any fixed key defines a permutation on the message space. We call such an encryption scheme PRP-CCA secure if the adversary cannot distinguish the permutation induced by the encryption function with a randomly generated key from a randomly chosen permutation over the same message space.

For *asymmetric CPA secure encryption*, we require that even with access to the public key no adversary can distinguish the encryptions of two self-chosen messages.

For *asymmetric CCA2 secure encryption*, we require the same as in CPA and additionally allow the adversary to query the decryption of (other) ciphertexts under the same key before and after her choice of messages.

A *rerandomizable CPA secure asymmetric encryption* scheme further includes a rerandomization algorithm. For that, we require that even with access to the public key no adversary can distinguish the rerandomization of a ciphertext from a fresh encryption of a random message.

A *key-private asymmetric encryption* scheme [15] additionally ensures that the adversary cannot distinguish which of two public keys was used to encrypt a message.

2.2.2 Authentication

For our constructions we use a SUF-CMA [28] (Strong Existential Unforgeability under Chosen Message Attack) secure message authentication code (MAC). A MAC scheme consists of a key generation, a signature generation and a signature verification algorithm. For correctness, any signature that is generated with a correctly generated key has to return true when it is verified, i.e. has to be valid. For security, we require SUF-CMA: The adversary can create a valid message-signature pair only with negligible probability, even though she gets to query signatures to messages that she chooses and she can reuse the same message as she queried. If she reuses the message of her query, she however needs to modify the signature of the query answer (as otherwise the "creation" is trivial).

2.2.3 Updatable Encryption

While we apply Updatable Encryption (UE) in the context of AC, UE originally [27, 89] targets the scenario of securely outsourcing data to a semi-trusted cloud server. To enable efficient key rotation, i.e., updating the stored ciphertexts to a freshly chosen key, UE schemes allow the generation of update tokens based on the old and new key. The generation of the token can be done independently of the ciphertext to be updated. Given such a token the server can autonomously lift a ciphertext encrypted with the old key to a ciphertext encrypted with the new key. Of course, the token itself may not leak information about the keys to the cloud server. In the following, we recapitulate the security notions of an UE scheme from [89].

UP-IND-RCCA Security. UP-IND-RCCA (Updatable Encryption Indistingishability under Replayable CCA) is conceptually similar to CCA2 above, except that the decryption queries for *all* ciphertexts that include one of the adversarially chosen messages are ignored. This change is necessary as the updating of ciphertexts in the corresponding scheme is non-deterministic and hence the adversary can apply the token multiple times to recover different ciphertexts for the same message. Additionally, UP-IND-RCCA allows the adversary to query re-encryptions and the current or earlier secret keys and tokens. Trivial wins, by corrupting keys and tokens that allow to decrypt the challenge ciphertext, are excluded.

Perfect Re-encryption. Intuitively, perfect re-encryption demands that fresh and re-encrypted ciphertexts are indistinguishable.

Plaintext Integrity UP-INT-PTXT. Plaintext integrity demands that the adversary cannot produce a ciphertext that decrypts to a message for which she does not trivially know an encryption, e.g. because she corrupted the corresponding keys or tokens earlier.

2.2.4 SNARGs

A succinct non-interactive argument (SNARG) [24] is a kind of Zero-Knowledge (ZK) proof, which proves a statement without revealing more information than is included in the statement (zero-knowlegde). More precisely, a SNARG is used by the prover to convince a verifier with the help of a witness from the fact that the statement is in the specified language. For this, a SNARG scheme consists of a key generation, a prove, a verify and a simulation algorithm. With the output of the key generation, the prove algorithm is used to generate an argument. This argument is later verified by the second party with the verify algorithm. SNARGs are succinct, complete, sound and zero-knowledge [77]. For our construction we specifically require simulation-soundness.

Succinctness intuitively requires the length of the arguments and the runtime of the verification to be short.

(Perfect) Completeness requires that *any* argument for a statement of the language generated by an honest execution of the protocol with a correct witness results in a successful verification, i.e. is a valid SNARG. Note that this is "perfect" in the sense that an honest execution with correct witness is *always* valid.

(Perfect) Zero-Knowledge requires that the outputs of the prove algorithm that has knowledge of the witness are identically distributed to the outputs of the simulation algorithm that does not know the witness. Hence, the SNARG does not leak *any* information about the witness ("perfect").

Simulation-Soundness requires that no adversary can with non-negligible probability generate a valid SNARG for a statement that is not in the language, even if the adversary has access to simulations of SNARGs for other statements of this language.

2.3. Privacy Goal Definitions

While concepts like k-anonymity [149], l-diversity [108], t-closeness [105] and various logic- and process calculus-based approaches [150, 83, 79, 139, 147, 23] have been discussed for the formalization of privacy goals, they come with the drawback of known attacks, are impractical to use in security proofs or simply target a different domain and are difficult to map to AC. Thus, currently the majority of ACNs, as well as this thesis, rely on differential privacy, indistinguishability games and ideal functionalities.

Additionally, we started to relate logic-based approaches to our indistinguishability game based approach and point the interested reader to our work at WPES 2021 [100].

2.3.1 Differential Privacy

For database privacy protection, the notion of differential privacy (DP) [60] is well accepted. Intuitively, an algorithm is differentially private if its output for any two neighboring databases is "very close". Neighboring databases are most commonly defined to differ only in one attribute of a single user. The "closeness" definition of the outputs depends on the version of differential privacy. We will use $(\epsilon, \delta) - DP$, which is defined as:
Definition 1 ([60])**.** *For any two data sets that differ on only one row, the respective output random variables (query responses) τ and τ‘ satisfy for all sets S of responses:*

$$\Pr[\tau \in S] \leq \exp(\epsilon) \times \Pr[\tau‘ \in S] + \delta$$

Thus an adversary's certainty of the estimated input is limited (as parameterized with ϵ) except for very few cases (corresponding to the probability δ).

2.3.2 Indistinguishability Games

Indistinguishability-based notions are well-known from cryptographic notions [75] (as CPA above) and follow a similar basic idea as DP: Any two cases that only differ in protected information have to be indistinguishable. To define this, the notions make use of a game between an adversary and a challenger: The adversary chooses and sends two options (e.g. two plaintexts in the CPA game) to the challenger. The challenger randomly picks one of them and returns the adversarial observations (e.g. the encryption of the chosen plaintext in the CPA game). Based on these observations the adversary has to guess which of the options the challenger used. The adversary wins the game if the guess was correct. The notion is achieved, if no probabilistic polynomial time (PPT) algorithm exists that in the role of the adversary wins with a probability that is non-negligibly higher than random guessing.

2.3.3 Ideal Functionalities in the Universal Composability Framework

An *ideal functionality* in the universal composability (UC) framework [32] is an abstraction of a real protocol that expresses the security and privacy properties as required for 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 that she would not learn from attacks on the ideal functionality.

While these abstractions are a neat simplification for the protocol analysis, they do not *directly* state the precise, offered protection, but instead only implicitly define it via the ideal protocol. The achieved protection thus still needs to be derived from an ideal functionality via analysis.

3. Privacy Notions for Anonymous Communication

In this chapter we introduce our privacy goal definitions as well as their relations, resolve naming conflicts in related works and present an initial investigation of building blocks to achieve the notions. Large parts of the results in this chapter have been published at PETS 2019 [93] (and in the corresponding extended version [92]).

3.1. Overview

We start with a use case as example and discuss the corresponding implicit privacy goal, to then introduce the idea of the related indistinguishability game. We show how such a game works and what it means for a protocol to be secure according to this goal. Furthermore, by adopting the game we sketch how privacy goals can be formalized as notions and provide an intuition for the relations between different goals.

EXAMPLE: *Alice is a citizen of a repressive regime and engaged with a resistance group. Despite the regime's sanctions on distributing critical content, Alice wants to publish her latest critical findings.* A vanilla encryption scheme would reduce Alice's potential audience and thus does not solve her problem. Hence, she needs to hide the link between the message and herself as the sender. We call this goal Sender-Message Unlinkability.[1]

First attempt. We start by presenting an easy game, that at first glance looks like the correct formalization for the goal of the example, but turns out to model an even stronger goal.

For Alice's safety, the regime should not suspect her of being the sender of a compromising message, otherwise she risks persecution. Thus, we need to show for the applied protection measure, that compared to any other sender, it is not more probable that Alice is the sender of this message. We analyze the worst case: in a group of users, let Charlie be the user for whom the probability of being the sender differs most from Alice's probability. If even these two are too close to distinguish, Alice is safe, since all other probabilities are closer. Hence, the regime cannot even exclude a single user from its suspects.

We abstract this idea into a game[2], where the adversary aims to distinguish two "worlds" or scenarios. These may only differ in the properties the protocol is required to protect, but within these restrictions the adversary can choose freely. This includes especially the worst case that is easiest for her to distinguish (e.g. in one scenario Alice sends the message, in the other Charlie). Fig. 3.1 shows the game in more detail.

What the adversary can observe in Step 4 depends on her capabilities and area of control. A weak adversary may only receive a message from somewhere, or discover it on a bulletin board. However, a stronger adversary could e.g. also observe the activity on the Internet uplinks of some parties.

The adversary wins the game if she guesses the correct scenario. Her goal is to devise a strategy

[1]Usually this is called sender anonymity. However, the term "sender anonymity" is overloaded and sometimes also used with a slightly different meaning. As the message should not be linkable to the sender for this goal, we instead refer to it as "Sender-Message Unlinkability".

[2]Similar to indistinguishability games in cryptology [75] and related works on anonymity[11, 26, 82, 72].

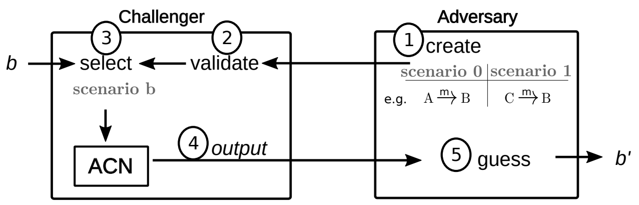


Figure 3.1: Steps of the sample game: **1)** adversary picks two scenarios; **2)** challenger checks if scenarios only differ in senders; **3)** based on random bit b the challenger inputs a scenario into the ACN; **4)** adversary observes execution; **5)** adversary outputs 'guess' as to which scenario was executed

that allows her to win the game repeatedly with a probability higher than random guessing. With this strategy she learned some information that is supposed to be protected, since everything else was identical in both scenarios. In our example this information is the sender (e.g. that Alice is more probable the sender of the message than Charlie). Hence, we say that, if the adversary can find such a strategy, we do not consider the analyzed protocol secure regarding the respective privacy goal.

Why the modeled goal is stronger than necessary for our example. As argued, a protocol achieving this goal would help Alice in her use case. However, if an adversary learns who is sending *any* message with real information (i.e. no random bits/dummy traffic), she can distinguish both scenarios and wins the game. The ACN thus needs to hide the sending activity (the adversary does not know if a certain possible sender was active or not) and we call the goal that we modeled so far *Sender Unobservability.* For Alice's use case, we only require Sender-Message Unlinkability and a protocol that hides the information of who sent a message within a set of active senders is good enough. As an example, consider the following two scenarios: (1) Alice and Bob send messages (2) Charlie and Dave send messages. If the adversary can learn the active senders, she can distinguish the scenarios and win the game for Sender Unobservability. However, if she only learns the set of active senders, she may still not know who of the two active senders in the played scenario actually sent the critical message.

Correcting the formalization. We can adjust the game of Fig. 3.1 to model Sender-Message Unlinkability. We desire that the only information about the communications that differs between the scenarios is who is sending which message. Thus, we allow the adversary to pick scenarios that differ in the senders, but not in the activity of the senders, i.e. the number of messages each active sender sends. This means, we change what the adversary is allowed to submit in Step 1 and what the challenger checks in Step 2. So, if the adversary now wants to use Alice and Charlie, she has to use both in both scenarios, e.g. (1) Alice sends the critical message, Charlie a benign message and (2) Charlie sends the critical message, Alice the benign message. Hence, given an ACN where this game cannot be won, the adversary is not able to distinguish whether Alice or another active user sent the critical message. The adversary might learn, e.g. that someone sent a critical message and the identities of all active senders (here that Alice and Charlie are active senders). However, since none of this is sanctioned in the above example, Alice is safe, and we say such an ACN provides Sender-Message Unlinkability.

Lessons learned. Depending on the formalized privacy goal (e.g. Sender Unobservability) the scenarios are allowed to differ in certain properties of the communications (e.g. the active senders) as we have illustrated in two exemple games. Following the standard in cryptology, we use the term *privacy notion*, to describe such a formalized privacy goal that defines properties to be hidden from the adversary.

Further, the games used to prove the privacy notions only differ in how scenarios can be chosen by the adversary and hence what is checked by the challenger. This also holds for all other privacy notions; they all define certain properties of the communication to be private and other properties that can leak to the adversary. Therefore, their respective games are structurally identical and can be abstracted to define one general game, whose instantiations represent notions. We explain and define this general game in Section 3.2. We then define the properties (e.g. that the set of active senders can change) in Section 3.3 and build notions (e.g. Sender Unobservability) upon them in Section 3.4.

Additionally, we already presented the intuition that Sender Unobservability is stronger than Sender-Message Unlinkability. This is not only true for this example, in fact we prove: every protocol achieving Sender Unobservability also achieves Sender-Message Unlinkability. Intuitively, if whether Alice is an active sender or not is hidden, whether she sent a certain message or not is also hidden. We will prove relations between our privacy notions in Section 3.5 and show that the presented relations (depicted in Figure 3.3) are complete. Further, we detail the relation to prior work in 3.6 and conduct an investigation on basic techniques to reach our notions against the GPA in Section 3.7. Finally, we discuss our lessons learned in Section 3.8..

As additional material, we argue our choice of notions in Appendix A.6 and detail extensions in Appendix A.1. Finally, we give quick instructions for a proof based on our notions in Appendix A.7.

3.2. Our Game model

Our goal in formalizing the notions as a game is to analyze a given ACN protocol w.r.t. to a notion, i.e. the game is a tool to investigate if an adversary can distinguish two self-chosen, notion-compliant scenarios. Scenarios are sequences of communications. A *communication* is described by its sender, receiver, message and auxiliary information (e.g. session identifiers) or the empty communication, signaling that nobody wants to communicate at this point. Some protocols might restrict the information flow to the adversary to only happen at specific points in the execution of the protocol, e.g. because a component of the ACN processes a batch of communications before it outputs statistics about them. Therefore, we introduce *batches* as a sequence of communications, which is processed as a unit before the adversary observes anything[3]. When this is not needed, batches can always be replaced with single communications.

As explained in Section 3.1, we do not need to define a completely new game for every privacy goal, since notions only vary in the difference between the alternative scenarios chosen by the adversary. Hence, for a given ACN and notion, our general game is simply instantiated with a model of the ACN, which we call the protocol model, and the notion. The protocol model accepts a sequence of communications as input. Similar to the real implementations the outputs of the protocol model are the observations the real adversary can make. Note, the adversaries in the game and the real world have the same capabilities, but differ in their aims: while the real world adversary aims to find out something about the users of the system, the game adversary merely aims to distinguish the two scenarios she has constructed herself.

In the simplest version of the game, the adversary constructs two scenarios, which are just two batches of communications and sends them to the challenger. The challenger checks that the batches are compliant with the notion. If so, the challenger tosses a fair coin to randomly decide which of the two batches it executes with the protocol model. The protocol model's output is returned to the game adversary. Based on this information, the game adversary makes a guess about the outcome of the coin toss.

We extend this simple version of the game, with three basic extensions. Firstly, we allow the game adversary to send two batches to the challenger not only once, but multiple times. However, the challenger performs a single coin flip and sticks to this scenario for this game, i.e. it always selects the batches corresponding to the initial coin flip. This allows analyzing for adversaries, that are able to base their next actions in the attack on the observations they made previously.

[3]We use the word batch to designate an accumulation of communications. Besides this similarity, it is not related to batch mixes.

Note that although the adversary decides on all the communications that happen in the alternative scenarios, she does not learn secret keys or randomness unless the user is corrupted. We thus, secondly, allow for user (a sender or receiver) corruption, i.e. the adversary learns the user's momentary internal state, by sending corrupt queries to the challenger. This allows to add several options for different corruption models to the privacy goals.

Thirdly, we allow the adversary to send protocol queries to unfetter our general game from the concrete adversary model. These queries are only a theoretical formalization to reflect what information the adversary gets and what influence she can exercise. These protocol query messages are usually sent to the protocol model without any changes by the challenger. The protocol model covers the adversary to ensure that everything the real world adversary can do is possible in the game with some query message. For example, protocol query messages can be used to add or remove nodes from the ACN by sending the appropriate message.

As introduced in Section 3.1, we say that an adversary has an advantage in winning the game, if she guesses the challenger-selected scenario correctly with a higher probability than random guessing. A protocol achieves a certain privacy goal, if an adversary has at most a negligible advantage in winning the game.

Formalization

In this subsection, we formalize the game model to conform to the above explanation.

We use Π to denote the analyzed *ACN protocol model*, Ch for the challenger and $\mathcal{A}$ for the adversary, which is a PPT algorithm. Additionally, we use X as a placeholder for the specific notion, e.g. Sender Unobservability, if we explain or define something for all the notions. A *communication* r in Π is represented by a tuple (u, u', m, aux) with a sender u, a receiver u', a message m, and auxiliary information aux (e.g. session identifiers). If no communication occurs, we represent this with $\Diamond$. Communications are clustered into *batches* $\underline{r}_b = (r_{b_1}, \dots, r_{b_l})$, with r_{b_i} being the i-th communication of batch $\underline{r}_b$. Note that we use $\underline{r}$ (underlined) to identify batches and r (no underline) for single communications. Batches in turn are clustered into *scenarios*; the first scenario is $(\underline{r}_{0_1}, \dots, \underline{r}_{0_k})$. All symbols are summarized in Tables A.2 – A.4 in Appendix A.4.

Simple Game.

1. Ch randomly picks challenge bit b.
2. $\mathcal{A}$ sends a batch query, containing $\underline{r}_0$ and $\underline{r}_1$, to Ch.
3. Ch checks if the query is valid, i.e. both batches differ only in information that is supposed to be protected according to the analyzed notion X.
4. If the query is valid, Ch inputs $\underline{r}_b$, i.e. the batch corresponding to b, to Π.
5. Π's output $\Pi(\underline{r}_b)$ is handed to $\mathcal{A}$.
6. After processing the information, $\mathcal{A}$ outputs her guess g for b.

Extensions. As explained above, there are useful extensions that we make to the simple game:

Multiple Batches Steps 2-5 can be repeated.

User corruption Instead of Step 2, $\mathcal{A}$ can also decide to issue a corrupt query specifying a user u and receive u's internal state as output. This might change Π's state, lead to different behavior of Π in following queries or yield a higher advantage in guessing than before.

Other parts of the adversary model Instead of Step 2, $\mathcal{A}$ can also decide to issue a protocol query, containing an input specific to Π and receive Π's output to it (e.g. the internal state of a router that is corrupted in this moment). This might change Π's state.

Achieving notion X. Intuitively, a protocol Π achieves a notion X if any possible adversary has at most negligible advantage in winning the game. To formalize the informal understanding of Π achieving goal X, we need the following denotation. $Ch(\Pi, X, c, b)$ is the challenger algorithm instantiated with protocol model Π and notion X that is allowing for at most c challenge row, i.e. communications differing in the scenarios. $\Pr[g = \langle \mathcal{A} \mid Ch(\Pi, X, c, b)\rangle]$ describes the probability that $\mathcal{A}$ outputs g, when Ch instantiated as above chose the challenge bit as b. With this probability, achieving a notion translates to Definition 2.

Definition 2 (Achieving a notion X)**.** *An ACN Protocol* Π *achieves X, iff for all PPT algorithms $\mathcal{A}$ there exists a negligible δ such that*

$$\left| \; Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 0)\rangle] - Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 1)\rangle] \; \right| \leq \delta.$$

We use a variable δ, which is referred to as negligible, as an abbreviation when we actually mean a function $\delta(\kappa)$ that is negligible in a security parameter κ. We discuss the equivalence to similar definitions and a relaxed differential privacy based advantage definition in Appendix A.5.1.

3.3. Protected Properties

We define properties to specify which information about the communication is allowed to be disclosed to the adversary, and which must be protected to achieve a privacy notion, as mentioned in Section 3.1. We distinguish between simple and complex properties. Simple properties can be defined with the basic game model already introduced, while complex properties require some extensions to the basic model.

3.3.1 Simple Properties

We summarize the informal meaning of all simple properties in Table 3.1 and introduce them in this section. We note that our selection and definition of simple properties is influenced by previous works [11, 26, 82, 72] and we discuss the relations in detail later in Section 3.6.1.

Assume an ACN aims to hide the message but discloses message lengths to observers. For this case, we specify the property ($|M|$) that the message length must not differ between the two scenarios, as this information must not help the adversary to distinguish which scenario the challenger chose to play.

Next, we might want an ACN to protect the identity of a sender, as well as any information about who sent a message, but deliberately disclose which messages are received by which receiver, who the receivers are, and potentially other auxiliary information. We hence specify a property (E_S) where only the senders differ between the two scenarios, to ensure that the adversary in our game can only win by identifying senders. In case the protection of the receiver identities or messages is required, the same can be defined for receivers (E_R) or messages (E_M).

Further, we might want the ACN to protect senders and also the messages; leaving the receiver and auxiliary information to be disclosed to the adversary. This is achieved by specifying a property where only senders and messages differ between the two scenarios and everything else remains equal (E_{SM}). Again, the same can be specified for receivers and messages (E_{RM}) or senders and receivers (E_{SR}).

Lastly, ACNs might allow the adversary to learn whether a real message is sent or even how many messages are sent. We specify a property ($\not\Diamond$) that requires real communications in both scenarios, i.e. it never happens that nothing is sent in one scenario but something is sent in the other. We ensure this by not allowing the empty communication ($\diamond$).

Table 3.1: Our[4]simple properties; information about communications that may be required to remain private

Symbol	Description	Translation to Game
$\|M\|$	Message Length	Messages in the two scenarios always have the same length.
E_S	Everything but Senders	Everything except the senders is identical in both scenarios.
E_R/E_M	Everything but Receivers/Messages	Analogous
E_{SM}	Everything but Senders and Messages	Everything except the senders and messages is identical in both scenarios.
E_{RM}/E_{SR}	Analogous	Analogous
$\not\Diamond$	Something is sent	In every communication something must be sent ($\Diamond$ not allowed).
$\aleph$	Nothing	Nothing will be checked; always true.
U/U'	Active Senders/Receivers	Who sends/receives is equal for both scenarios.
Q/Q'	Sender/Receiver Frequencies	Which sender/receiver sends/receives how often is equal for both scenarios.
$\|U\|/\|U'\|$	Number of Senders/Receivers	How many senders/receivers communicate is equal for both scenarios.
P/P'	Message Partitioning per Sender/Receiver	Which messages are sent/received from the same sender/receiver is equal for both scenarios.
H/H'	Sender/Receiver Frequency Histograms	How many senders/receivers send/receive how often is equal for both scenarios.

However, a very ambitious privacy goal might even require that the adversary learns no information about the communication at all ($\aleph$). In this case, we allow any two scenarios and check nothing.

Formalizing those Simple Properties. In the following definition all simple properties mentioned so far are formally defined. Therefore, we use $\top$ as symbol for the statement that is always true.

Definition 3 (Properties $|M|$, E_S, E_{SM}, $\not\Diamond$, $\aleph$)**.** *Let the checked batches be $\underline{r_0}, \underline{r_1}$, which include the communications $r_{0j} \in \{(u_{0_j}, u'_{0_j}, m_{0_j}, aux_{0_j}), \diamond\}$ and $r_{1j} \in \{(u_{1_j}, u'_{1_j}, m_{1_j}, aux_{1_j}), \diamond\}$ with $j \in \{1, \dots l\}$. We say the following properties are met, iff for all $j \in \{1, \dots l\}$:*

$$|M| : |m_{0j}| = |m_{1j}|$$

$$\begin{aligned} E_S &: r_{1j} = (\mathbf{u_{1_j}}, u'_{0_j}, m_{0_j}, aux_{0_j}) \\ E_R &: r_{1j} = (u_{0_j}, \mathbf{u'_{1_j}}, m_{0_j}, aux_{0_j}) \\ E_M &: r_{1j} = (u_{0_j}, u'_{0_j}, \mathbf{m_{1_j}}, aux_{0_j}) \end{aligned}$$

$$\begin{aligned} E_{SM} &: r_{1j} = (\mathbf{u_{1_j}}, u'_{0_j}, \mathbf{m_{1_j}}, aux_{0_j}) \\ E_{RM} &: r_{1j} = (u_{0_j}, \mathbf{u'_{1_j}}, \mathbf{m_{1_j}}, aux_{0_j}) \\ E_{SR} &: r_{1j} = (\mathbf{u_{1_j}}, \mathbf{u'_{1_j}}, m_{0_j}, aux_{0_j}) \end{aligned}$$

$$\begin{aligned} \not\Diamond &: \Diamond \notin \underline{r_0} \land \Diamond \notin \underline{r_1} \\ \aleph &: \top \end{aligned}$$

[4]We note that protections similar to E_S have been defined by [11, 26, 82] and while our definitions take another approach $U, |U|, Q, H$ define the corresponding properties as in [26] and [72].

More Simple Properties: Active Users, Frequencies. The properties of Definition 3 are important to formalize privacy, but are by themselves not sufficient. Take the ACN Tor as an example: While the set of active senders is trivially known to their ISPs and the guard nodes, we still require that the senders are unlinkable with the messages they are sending (and their receivers). Similarly, the sending (receiving) frequency of a party may be important and is not formalized yet. To formalize these properties, we use sets that capture which user sent which messages in a certain period, i.e. a batch of communications (and similarly sets to capture which user received which messages). Note that we use primes ($'$) for the corresponding sets and properties of the receivers.

Definition 4 (Sender-Message Linking)**.** *We define the sender-message linkings for scenario b (L'_{b_i} as the receiver-message linkings similarly) as:*

$$L_{b_i} := \{(u, \{m_1, ..., m_h\}) \mid u \text{ sent messages } m_1, \ldots, m_h \text{ in batch } i\}.$$

The sets from Definition 4 allow easy identification of who an active sender in this batch was and how often each sent something:

Definition 5 (Active Sender Set, Frequency Set)**.** *Let the current batch be the k-th one. For $b \in \{0, 1\}$ U_b, Q_b (U'_b, Q'_b for L'_b) are defined as:*

$$U_b := \{u \mid (u, M) \in L_{b_k}\}$$
$$Q_b := \{(u, n) \mid (u, M) \in L_{b_k} \land |M| = n\}$$

Recall that we currently define properties for ACNs that allow the adversary to learn which senders are active at different times, or the number of messages they send during some periods, while hiding some other properties (e.g. which messages they have sent). Hence, with the respective sets for active users and user frequencies defined, we only need to request that they are equal in both scenarios:

Definition 6 (Properties U, Q, $|U|$)**.** *We say that the properties $U, Q, |U|$ ($U', Q', |U'|$ analogous) are met, iff:*

$$U : U_0 = U_1 \qquad Q : Q_0 = Q_1 \qquad |U| : |U_0| = |U_1|$$

More Simple Properties: Message Partitions, Histograms. Other interesting properties are which messages came from a given sender and how many senders sent how many messages. If the adversary knows which messages are sent from the same sender, e.g. because of a pseudonym, she might be able to combine information from them all to identify the sender. If she knows how many senders sent how many messages, she knows the sender activity and hence can make conclusions about the nature of the senders.

As before, we introduce auxiliary variables to formally define these two properties. We use the multiset $M_{b,I}$ to denote the collection of messages that has been sent by the same sender (e.g. linked by a shared pseudonym) in a set of batches, and the multiset $M_{b,I,n}$ to denote the union of all these sets of cardinality n. The equality of the properties in the two scenarios must pertain throughout all comparable batches in the scenarios. If this were not true, the inequality would help the adversary to distinguish the scenarios without learning the protected information e.g. identifying the sender.

Definition 7 (Multi-Batch-Message Linkings)**.** *Let the current batch be the k-th, $\mathcal{K} := \{1, \ldots, k\}$, $\mathcal{P}(\mathcal{K})$ the power set of $\mathcal{K}$ and $\mathcal{U}$ the set of all possible senders ($\mathcal{U}'$ receivers). For $b \in \{0, 1\}$ and $I \in \mathcal{P}(\mathcal{K})$: We define ($M'_{b,I}, M'_{b,I,n}$ for L'_{b_i}) the*

- *multi-batch-message-sender linking: $M_{b,I} := \cup_{u \in \mathcal{U}} \{\cup_{i \in I} \{M | (u, M) \in L_{b_i}\}\}$ and the*
- *cardinality restricted multi-batch-message-sender linking: $M_{b,I,n} := \{M \in M_{b,I} \mid |M| = n\}$.*

As before, we define auxiliary variables capturing the information that we want to be equal in both scenarios: We define tuples specifying which messages are sent from the same user for any set of batches (Message Partition P_b) and how many users sent how many messages for any set of batches (Histogram H_b). Therefore, we use a slightly unusual notation: For any set Z, we use $(Z_i)_{i\in\{1,\dots,k\}}$ to denote the sequence $(Z_1, Z_2, \dots, Z_k)$ and $\overrightarrow{\mathcal{P}}(Z)$ to denote a sorted sequence of the elements of the power set[5] of Z.

Definition 8 (Message partitions, Histograms)**.** *Consider the k-th batch, $\mathcal{K} := \{1, \dots, k\}$. For $b \in \{0,1\}$ P_b, H_b (P'_b, H'_b analogous) are defined as:*

$$P_b := (M_{b,I})_{I\in\overrightarrow{\mathcal{P}}(\mathcal{K})}$$
$$H_b := (\{(n,i) \mid i = |M_{b,I,n}|\})_{I\in\overrightarrow{\mathcal{P}}(\mathcal{K})}$$

Further, we say that properties P, H (P', H' analogous) are met, iff:

$$P : P_0 = P_1 \qquad H : H_0 = H_1$$

3.3.2 Complex Properties

So far, we have defined various properties to protect senders, messages, receivers, their activity, frequency and the grouping of messages. However, this is not sufficient to formalize several relevant privacy goals, and we must hence introduce complex properties. We note that our selection and definition of complex properties is influenced by previous work [11] and we discuss the relations in detail later in Section 3.6.1.

Learning Sender and Receiver. Consider that one aims to hide which sender is communicating with which receiver. Early ACNs like classical Mix-Nets [42], and also Tor [58], already used this goal. Therefore, we want the adversary to win the game only if she links the sender and receiver of a message.

An intuitive solution may be to model this goal by allowing the adversary to pick different senders and receivers (E_{SR}) in both scenarios (see Fig. C.1 (a) for an example). This, however, does not actually model the privacy goal: by identifying only the sender or only the receiver of the communication, the game adversary could tell which scenario was chosen by the challenger. We hence must extend the simple properties and introduce scenario *instances* to model dependencies.

Scenario instances. We now require the adversary to give alternative instances for both scenarios (Fig. C.1 (b)). The challenger chooses the scenario according to the challenge bit, which is picked randomly for every game, and the instance according to the instance bit, which is picked randomly for every challenge.

Formally, we replace steps 2–5 of the game with the following steps:

2. $\mathcal{A}$ sends a batch query, containing $\underline{r}_0^0$, $\underline{r}_0^1$, $\underline{r}_1^0$and $\underline{r}_1^1$ to Ch.

3. Ch checks if the query is valid according to the analyzed notion X.

4. If the query is valid and Ch has not already picked an instance bit a for this challenge, Ch picks $a \in \{0,1\}$ randomly and independent of b. Then it inputs the batch corresponding to b and a to Π.

5. Π's output $\Pi(\underline{r}_b^a)$ is forwarded to $\mathcal{A}$.

[5]For brevity we use $\in$ to iterate through a sequence.

This allows us to model the goal that the adversary is not allowed to learn the sender and receiver: We allow the adversary to pick two sender-receiver pairs, which she uses as instances for the first scenario. The mixed sender-receiver pairs must then be provided as instances for the second scenario (see Fig. C.1 (b)). We thus force the game adversary to provide alternative assignments for each scenario. This way she cannot abuse the model to win the game by identifying only the sender or the receiver. We call this property *Random Sender Receiver* R_{SR}.

This complex property is still not sufficient to model the situation in, for example, Tor:

The adversary can distinguish the scenarios without learning who sent to whom, just by learning which senders and which receivers are active. Hence, we further restrict the adversary picking instances where both senders and both receivers are active by defining the property *Mix Sender Receiver* M_{SR}. Here, the adversary picks two instances for $b = 0$ where her chosen sender-receiver pairs communicate, and two for $b = 1$ where the mixed sender-receiver pairs communicate. The two instances simply swap the order in which the pairs communicate (Fig. C.1 (c)). This way, we force the adversary to provide alternative assignments for each scenario where both suspected senders and both suspected receivers are active. This combination prevents the adversary from winning the game without learning the information that the real system is actually supposed to protect, i.e. the sender-receiver pair.

a)

scenario 0	scenario 1
A $\longrightarrow$ B	C $\longrightarrow$ D

b)

	scenario 0	scenario 1
instance 0	A $\longrightarrow$ B	A $\longrightarrow$ D
instance 1	C $\longrightarrow$ D	C $\longrightarrow$ B

c)

	scenario 0	scenario 1
instance 0	A $\longrightarrow$ B C $\longrightarrow$ D	A $\longrightarrow$ D C $\longrightarrow$ B
instance 1	C $\longrightarrow$ D A $\longrightarrow$ B	C $\longrightarrow$ B A $\longrightarrow$ D

Figure 3.2: Examples showing the general structure of communications that differ in both scenarios: a) Naive, but incorrect b) Random Sender Receiver R_{SR} c) Mixed Sender Receiver M_{SR}

Defining Complex Properties. To simplify the formal definition of complex properties, we introduce *challenge rows.* A challenge row is a pair of communications with the same index that differ in the two scenarios (e.g. r_{0_j}, r_{1_j} with index j). For complex properties, the challenger only checks the differences of the challenge rows in the two scenarios.

Definition 9 (Properties R_{SR}, M_{SR})**.** *Let the given batches be $\underline{r}_b^a$ for instances $a \in \{0,1\}$ and scenarios $b \in \{0,1\}$,* CR *the set of challenge row indexes, $(u_0^a, {u'}_0^a)$ for both instances $a \in \{0,1\}$ be the sender-receiver-pairs of the first challenge row of the first scenario ($b = 0$) in this challenge. Random Sender Receiver R_{SR}, Mixed Sender Receiver M_{SR} ($R_{SM}, R_{RM}, M_{SM}, M_{RM}$ similarly) are met, iff:*

$$R_{SR}: \quad r^a_{0\,cr} = (\mathbf{u_0^a}, \mathbf{u'}_\mathbf{0}^\mathbf{a}, m^1_{0_{cr}}, aux^1_{0_{cr}}) \quad \wedge \quad r^a_{1\,cr} = (\mathbf{u_0^a}, \mathbf{u'}_\mathbf{0}^\mathbf{1-a}, m^1_{0_{cr}}, aux^1_{0_{cr}})$$
$$\forall cr \in \mathsf{CR}, a \in \{0,1\}$$

$$M_{SR}: \quad r^a_{0\,cr} = (\mathbf{u_0^a}, \mathbf{u'}_\mathbf{0}^\mathbf{a}, m^1_{0_{cr}}, aux^1_{0_{cr}}) \quad \wedge \quad r^a_{1\,cr} = (\mathbf{u_0^a}, \mathbf{u'}_\mathbf{0}^\mathbf{1-a}, m^1_{0_{cr}}, aux^1_{0_{cr}}) \ \wedge$$
$$r^a_{0\,cr+1} = (\mathbf{u_0^{1-a}}, \mathbf{u'}_\mathbf{0}^\mathbf{1-a}, m^1_{0_{cr}}, aux^1_{0_{cr}}) \quad \wedge \quad r^a_{1\,cr+1} = (\mathbf{u_0^{1-a}}, \mathbf{u'}_\mathbf{0}^\mathbf{a}, m^1_{0_{cr}}, aux^1_{0_{cr}})$$
$$\textit{for every second } cr \in \mathsf{CR}, a \in \{0,1\}$$

Linking message senders. A final common privacy goal that still cannot be covered is whether a pair of messages was sent by the same sender. This includes whether one sender was active twice or two different senders were active (Twice Sender Unobservability). Assume a real world adversary that can determine that the sender of two messages is the same entity. If subsequently she discovers the identity of the sender of one of the messages through a side channel, she can also link the second message to the same individual.

Stages. To model this goal, we need two scenarios (1) both messages are sent by the same sender, and (2) each message is sent by a different sender. Further, the adversary picks the messages for which she wants to decide whether they are sent from the same individual, and which other messages are sent between those two messages. Therefore, we add the concept of *stages* and ensure that only one sender sends in the challenge rows of Stage 1, and in Stage 2 either the same sender continues sending ($b = 0$) or another sender sends those messages ($b = 1$). This behavior is specified as the property *Twice Sender T_S*.

Definition 10 (Property T_S)**.** *Let the given batches be $\underline{r}_b^a$ for instances $a \in \{0,1\}$ and scenarios $b \in \{0,1\}$, x the current stage,* CR *the set of challenge row indexes, $(u_0^a, {u'}_0^a)$ for both instances $a \in \{0,1\}$ be the sender-receiver-pairs of the first challenge row of the first scenario ($b = 0$) in this challenge in Stage 1 and $(\tilde{u}_0^a, \tilde{u}_0'^a)$ the same pairs in Stage 2. Twice Sender T_S is met, iff (T_R similarly):*

$$\begin{aligned}
T_S:\quad & x = Stage1 \ \wedge \quad {r_0^a}_{cr} = (\mathbf{u_0^a}, {u'}_0^{\,0}, m^1_{0_{cr}}, aux^1_{0_{cr}}) \ \wedge \quad {r_1^a}_{cr} = (\mathbf{u_0^a}, {u'}_0^{\,0}, m^1_{0_{cr}}, aux^1_{0_{cr}}) \\
\vee \quad & x = Stage2 \ \wedge \quad {r_0^a}_{cr} = (\mathbf{u_0^a}, \tilde{u}_0'^{\,0}, m^1_{0_{cr}}, aux^1_{0_{cr}}) \ \wedge \quad {r_1^a}_{cr} = (\mathbf{u_0^{1-a}}, \tilde{u}_0'^{\,0}, m^1_{0_{cr}}, aux^1_{0_{cr}}) \\
& \forall cr \in \mathsf{CR}, a \in \{0,1\}
\end{aligned}$$

Hence, we need to facilitate distinct stages for notions with the complex properties T_S or T_R. Precisely, in Step 2 of the game, the adversary is additionally allowed to switch the stages.

This set of properties allows us to specify all privacy goals that have been suggested in literature[6] as privacy notions and additionally all that we consider important. It is of course difficult to claim completeness, as future ACNs may define diverging privacy goals and novel observable properties may be discovered.

3.4. Privacy Notions

Given the properties above, we can now set out to express intuitive privacy goals as formal privacy notions. We provide the definitions, explanation to our naming scheme as well as our clustering of notions and an overview of additional options that cover session, corruption, quantification and adversarial restrictions in this subsection. Additionally, we present one illustrative use case for each notion in Appendix A.6.

Notion Definition. We start by specifying Sender Unobservability as an example leading to a general definition of our privacy notions.

Recall the first game we defined in Section 3.1, which corresponds to Sender Unobservability ($S\overline{O}$ = S(ender) ¬ O(bservability)). There, in both scenarios something has to be sent, i.e. we need to specify that sending nothing is not allowed: $\not\Diamond$. Further, both scenarios can only differ in the senders, i.e. we also need the property that everything but the senders is equal: E_S. Hence, we define Sender Unobservability as $S\overline{O} := \not\Diamond \wedge E_S$.[7]

We define all other notions in the same way:
Definition 11 (Notions)**.** *Privacy notions are defined as a boolean expression of the properties according to Table 3.2.*

Modeling the notions as a game, the respective challenger verifies all properties (and the later introduced options) of the adversary's queries. A complete description of the challenger can be found in Appendix A.2. Further, an example of how the definitions can be represented by using a challenge specific state, which the challenger maintains, is shown in Appendix A.3.

Naming Scheme. We want to explain our naming scheme, which we summarize in Table 3.3. Our notions consider three dimensions: senders, messages and receivers. Each notion restricts the

[6]We note that protections similar to our properties R_{SR} and T_S have been defined by [11].
[7]Technically E_S already includes $\not\Diamond$. However, to make the differences to other notions more clear, we decide to mention both in the definition.

Table 3.2: Definition of the notions with $X' \in \{R\overline{O}, R\overline{O}-|U'|, R\overline{O}-H', R\overline{O}-P', RF\overline{L}, RF\overline{L}-H', RF\overline{L}-P', RM\overline{L}, RM\overline{L}-P'\}$. A description of the simple properties was given in Table 3.1.

Notion	**Properties**
$(SR)\overline{L}$	$\not\Diamond \wedge E_{SR} \wedge M_{SR}$
$(SR)\overline{O}$	$\not\Diamond \wedge E_{SR} \wedge R_{SR}$
$M\overline{O}$	$\not\Diamond \wedge E_M$
$M\overline{O}-\|M\|$	$\not\Diamond \wedge E_M \wedge \|M\|$
$M\overline{O}[M\overline{L}]$	$\not\Diamond \wedge Q \wedge Q'$
$\overline{O}$	$\not\Diamond$
$C\overline{O}$	$\aleph$
$S\overline{O}$	$\not\Diamond \wedge E_S$
$S\overline{O}-\|U\|$	$\not\Diamond \wedge E_S \wedge \|U\|$
$S\overline{O}-H$	$\not\Diamond \wedge E_S \wedge H$
$S\overline{O}-P$	$\not\Diamond \wedge E_S \wedge P$
$SF\overline{L}$	$\not\Diamond \wedge E_S \wedge U$
$SF\overline{L}-H$	$\not\Diamond \wedge E_S \wedge U \wedge H$
$SF\overline{L}-P$	$\not\Diamond \wedge E_S \wedge U \wedge P$
$SM\overline{L}$	$\not\Diamond \wedge E_S \wedge Q$
$SM\overline{L}-P$	$\not\Diamond \wedge E_S \wedge Q \wedge P$
$(2S)\overline{O}$	$\not\Diamond \wedge E_S \wedge T_S$
$R\overline{O}$ etc.	analogous
$S\overline{O}[M\overline{O}]$	$\not\Diamond \wedge E_{SM}$
$S\overline{O}[M\overline{O}-\|M\|]$	$\not\Diamond \wedge E_{SM} \wedge \|M\|$
$(SM)\overline{O}$	$\not\Diamond \wedge E_{SM} \wedge R_{SM}$
$(SM)\overline{L}$	$\not\Diamond \wedge E_{SM} \wedge M_{SM}$
$R\overline{O}[M\overline{O}-\|M\|]$ etc.	analogous
$S\overline{O}\{X'\}$	Properties of X', remove E_R
$R\overline{O}\{X\}$	analogous

amount of leakage on each of those dimensions. However, only dimensions that are to be protected are part of the notion name. We use $\overline{O}$, short for unobservability, whenever the set of existing items of this dimension cannot be leaked to the adversary. E.g. $S\overline{O}$ cannot be achieved if the set of senders U is leaked. Notions carrying $\overline{L}$, short for unlinkability, can leak U (for sender related notions), but not some other property related to the item. E.g. we use $SF\overline{L}$ if the frequency Q cannot be leaked and $SM\overline{L}$, if Q can be leaked, but not the sender-message relation. With a "$-Prop$" we signal that the property $Prop$ can additionally leak to the adversary. We distinguish those properties from U and Q used before as they give another leakage dimension (as illustrated later in the hierarchy). Further, we use parentheses as in $(SR)\overline{O}$ to symbolize that if not only one set, but both sets of senders and receivers (U and U') are learned the notion is broken. Analogously, in $(SR)\overline{L}$ both sets can be learned but the linking between sender and receiver cannot. For the last missing complex property, we use $(2S)\overline{L}$ to symbolize that two senders have to be linked to be the same identity to break this notion.

For readability we add some abbreviations: We use $\overline{O} = S\overline{O}R\overline{O}M\overline{O}$ to symbolize unobservability on all three types and we summarize the remaining types in $M\overline{O}(SM\overline{L}, RM\overline{L})$ to $M\overline{O}[M\overline{L}]$. $C\overline{O}$ symbolizes the notion in which nothing is allowed to leak. Further, we use curly brackets to symbolize that the message cannot be leaked $S\overline{O}\{X\} = S\overline{O}M\overline{O}X$ and we put the (in our understanding) non dominating part of the notion in brackets $S\overline{O}M\overline{O} = S\overline{O}[M\overline{O}]$.

Notion Groups. We chose to group our notions semantically. Our resulting clusters are shown as gray boxes in Figure 3.3. Horizontally, we categorize notions that focus on receiver or sender protection (Receiver Privacy Notions or Sender Privacy Notions, respectively) or treat both with the same level of importance (Impartial Notions). Inside those categories, we use clusters concerning the general leakage type: Both-side Unobservability means that neither senders, nor receivers or messages should be leaked. Both-side Message Unlinkability means that it should be possible to link neither senders nor receivers to messages. In Sender Observability, the sender of every communication can be known, but not the message she sends or to whom she sends (Receiver and Message Observability analogous). In Sender-Message Linkability, who sends which message can

Table 3.3: Naming Scheme

Usage	Explanation
$D \in \{S, R, M\}$	Dimension $\in$ {Sender, Receiver, Message}
Dimension D not mentioned	Dimension can leak
Dimension D mentioned	Protection for this dimension exists
$D\overline{O}$	not even the participating items regarding D leak,(e.g. $S\overline{O}$: not even U leaks)
$DF\overline{L}$	participating items regarding D can leak, but not which exists how often (e.g. $SF\overline{L}$: U leaks, but not Q)
$DM\overline{L}$	participating items regarding D and how often they exist can leak (e.g. $SM\overline{L}$: U, Q leaks)
$X - Prop$, $Prop \in \{\lvert U\rvert, H, P, \lvert U'\rvert, H', P', \lvert M\rvert\}$	like X but additionally Prop can leak
$(D_1D_2)\overline{O}$	uses $R_{D_1D_2}$; participating items regarding D_1, D_2 do not leak, (e.g. $(SR)\overline{O}$: R_{SR})
$(D_1D_2)\overline{L}$	uses $M_{D_1D_2}$; participating items regarding D_1, D_2 can leak, (e.g. $(SR)\overline{L}$: M_{SR})
$(2D)\overline{L}$	uses T_D; it can leak whether two participating item regarding D are the same, (e.g. $(2S)\overline{O}$: T_S)
$\overline{O}$	short for $S\overline{O}R\overline{O}M\overline{O}$
$M\overline{O}[M\overline{L}]$	short for $M\overline{O}(SM\overline{L}, RM\overline{L})$
$S\overline{O}\{X\}$	short for $S\overline{O}M\overline{O}X$
$D_1X_1[D_2X_2]$	D_1 is dominating dimension, usually D_1 has more freedom, i.e. X_2 is a weaker restriction than X_1
$C\overline{O}$	nothing can leak (not even the existence of any communication)

Table 3.4: Our[8] options for corruption and for corrupted communication

Symbol	Description
X	Default option: Adaptive corruption is allowed.
X_{c^-}	Static corruption: users can only get corrupted before the first batch query is sent.
X_{c^0}	No corruption of users is allowed.
X	Default option: Corrupted users not restricted.
$X_{c^{\text{sr}}}$	Corrupted users are not allowed to be chosen as senders or receivers.
X_{c^s}	Corrupted users are not allowed to be senders.
X_{c^r}	Corrupted users are not allowed to be receivers.
X_{c^e}	Corrupted users send/receive identical messages in both scenarios.

be known to the adversary (Receiver-Message and Sender-Receiver Linkability analogous).

Additional Options. Additionally to the properties, we define options. Options can be added to any notion and allow for a more precise mapping of real world protocols, aspects of the adversary model, or easier analysis by quantification. We note that our selection and definition of options is influenced by previous works [11, 82, 72] and we discuss the relations in detail later in Section 3.6.1. We detail and define all options in Appendix A.1 and give an overview here.

Sessions. Some ACN protocols, like e.g. Tor, use sessions. Sessions encapsulate sequences of communications from the same sender to the same receiver by using the same session identifier for them. In reality, the adversary might be able to observe the session identifiers but (in most cases) not to link them to a specific user. We thus introduce an additionally restricted notion X_s for any notion X, which similar to definitions in [11] ensures that the adversary does not already win by merely recognizing a session based on its identifier.

Corruption. Some adversary capabilities like user corruption imply additional checks our challenger has to do. As all properties are independent from corruption, we add corruption as an option, that can be more or less restricted as shown in Table 3.4.

[8]Similar restrictions as $X_{c^{\text{sr}}}$ and X_{c^s} have been used by [72],X_{c^-} by [11, 72], X_{c^0} by [82].

Quantification. For an easier analysis, we allow the quantification of notions in the options similar to [11]. This way a reduced number of challenge rows (challenge complexity) or of challenges (challenge cardinality) can be required. Appendix A.1.4 includes information on how results with low challenge cardinality imply results for higher challenge cardinalities.

Adversary Classes. We adapt the concept of adversary classes from Anoa [11]. The adversary model assumed in the protocol model can be further restricted by adding adversary classes, that filter the information from the adversary to the challenger and vice versa. Potentially many such adversary classes can be defined. Some of them allow for privacy guarantees based on a single challenge (see Appendix A.1.4 for details).

UC-Realizability. AnoA shows that, if a protocol Π UC-realizes an ideal functionality $\mathcal{F}$, which achieves $(c, \epsilon, \delta) - X$, Π also achieves $(c, \epsilon, \delta + \delta')$-X for a negligible δ'. As the proof is based on the (ϵ, δ)- differential privacy definition of achieving a notion and independent from our extensions to the AnoA framework, this result still holds (see Appendix A.5.4).

3.5. Hierarchy

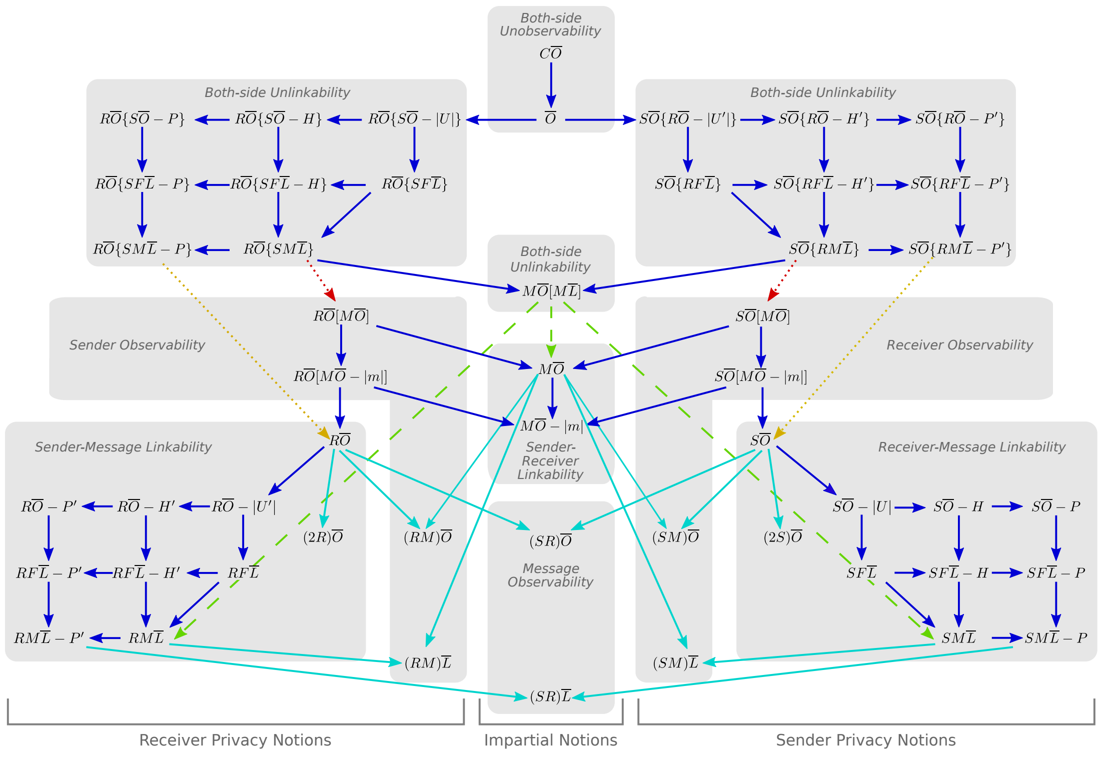


Figure 3.3: Our new hierarchy of privacy notions divided into sender, receiver and impartial notions and clustered by leakage type

$$\begin{tikzcd}
 & & X \arrow[r, phantom, "\Longleftrightarrow"] \arrow[dll] \arrow[dl] \arrow[d] & X_{CR_c} \arrow[d] \arrow[dr] & \\
X_s & X_{c^s} \arrow[dr] & X_{c^e} \arrow[d] & X_{c^r} \arrow[dl] & X_{c^-} \arrow[d] \\
 & & X_{c^{sr}} \arrow[rr] & & X_{c^0}
\end{tikzcd}$$

Figure 3.4: Additional implications for corruption and sessions

Next, we want to compare all notions and establish their hierarchy. To do this, for any pair of notions we analyze which one is stronger than, i.e. implies, the other. This means, any ACN achieving the stronger notion also achieves the weaker (implied) one. Our result is shown in Figure 3.3, where all arrow types represent implications, and is proven as Theorem 1 below. Further, obvious implications between every notion $S\overline{O}\{X\}$, $R\overline{O}\{X\}$ and X exist, since $S\overline{O}\{X\}$ only adds more possibilities to distinguish the scenarios. However, to avoid clutter we do not show them in Figure 3.3. Further, the same hierarchy exists between notions with the same session, corruption and quantification options. Additionally, we add a small hierarchy for the options that holds by definition in Figure 3.4.

Theorem 1. *The implications shown in Figure 3.3 hold.*

Proof. (See Appendix A.5.2 for details.) Generally, we[9] prove every implication $X_1 \Rightarrow X_2$ by an indirect proof of the following outline: Given an attack on X_2, we can construct an attack on X_1 with the same success. Assume a protocol has X_1, but not X_2. Because it does not achieve X_2, there exists a successful attack on X_2. However, this implies that there exists a successful attack on X_1 (we even know how to construct it). This contradicts that the protocol has X_1. Due to this construction in the proof the implications are transitive.

We use different arrow styles in Figure 3.3 to partition the implications into those with analogous proofs.

⇢ For the dashed, green implications $(c, \epsilon, \delta) - X_1 \Rightarrow (c, \epsilon, \delta) - X_2$ the attack on X_2 is also valid for X_1, because the restriction on the other dimensions (e.g. E_S) assures the same frequencies for the fixed dimensions (e.g. Q').

⇢ For the dotted, yellow implications $(c, \epsilon, \delta) - X_1 \Rightarrow (c, \epsilon, \delta) - X_2$ the attack on X_2 is also valid for X_1, because the restriction on the other dimensions (e.g. E_S) assures the same partitions for the fixed dimensions (e.g. P').

→ All dark blue implications $(c, \epsilon, \delta) - X_1 \Rightarrow (c, \epsilon, \delta) - X_2$ follow by definition of the properties.

⇢ For the dotted, red implications $(c, \epsilon, \delta) - X_1 \Rightarrow (c, \epsilon, \delta) - X_2$ the attack on X_2 is also valid for X_1, because the restriction on the other dimension (e.g. E_{RM}) assures the same frequency for the fixed dimension (e.g. Q).

→ For the cyan implications $(c, \epsilon, \delta) - X_1 \Rightarrow (c, \epsilon, 2\delta) - X_2$ we construct two attacks of which at least one has to be successful (with an adapted δ).

Additionally, the corrupt queries are not changed by the proposed constructions. Hence, the implications hold true between those notions as long as they have the same corruption options. Analogously sessions are not modified by the constructions and the same implications hold true between notions with equal session options. □

Additional implications based on corruption and sessions are shown in Figure 3.4. Most of them hold by definition. Only the equivalence with and without challenge row restriction per challenge is not so easy to see and proven in Theorem 12 of Appendix A.5.5.

So far we have proven that implications between notions exist. Further, we assure that the hierarchy is complete, i.e. that there exist no more implications between the notions of the hierarchy:

Theorem 2. *For all notions X_1 and X_2 of our hierarchy, where $X_1 \implies X_2$ is not proven or implied by transitivity, there exists an ACN protocol achieving X_1, but not X_2.*

Proof. Overview. We construct the protocol in the following way: Given a protocol Π that achieves X_1' (X_1 itself or a notion that implies X_1), let protocol Π' run Π and additionally output some information I. We argue that learning I does not lead to any advantage distinguishing the scenarios for X_1. Hence, Π' achieves X_1. We give an attack against X_2 where learning I allows to distinguish the scenarios. Hence, Π' does not achieve X_2. Further, we use the knowledge that

[9]In AnoA and Bohli's and Hevia's framework some of these implications are proved for their notions with the same approach.

$\implies$ is transitive[10] and give the systematic overview over all combinations and the corresponding proofs in Appendix A.5.5. $\square$

3.6. Relations to Prior Work

We compare our definitions with existing analysis frameworks and the goals of a proposed ACN.

3.6.1 Relation to Existing Analysis Frameworks

In this section, we introduce existing frameworks and point out to which of our notions their notions correspond. We argue that our framework includes all their assumptions and notions relevant for ACNs and thus provides a combined basis for an analysis of ACNs. For each framework, we first quickly give an idea why the properties and options match the notions of it and focus on how the concepts (like batches) relate later on. The resulting mapping is shown in Table 3.5, as well as partially illustrated in Figure A.1 of Appendix A.4 and reasoned below.

AnoA Framework. AnoA [11] builds its privacy notions on (ϵ, δ) differential privacy and compares them to their interpretation of the terminology paper of Pfitzmann and Hansen [126].

We derive properties that allow us to map AnoA's anonymity functions to our notions in the following way. AnoA's α_{SA} allows only one sender to change, the same is achieved with the combination of E_S and one challenge row. In AnoA's α_{RA} also the messages can differ, but have to have the same length, which we account for with using E_{RM} and $|M|$. AnoA's α_{REL} will either end in one of the given sender-receiver combinations been chosen ($b = 0$) or one of the mixed cases ($b = 1$). This is exact the same result as R_{SR} generates. For AnoA's α_{UL} either the same sender is used in both stages or each of the senders is used in one of the stages. This behavior is achieved by our property T_S. Although AnoA checks that the message length of the communication of both scenarios is equal, only the first message is used in any possible return result of α_{UL}. Hence, not checking the length and requiring the messages to be the same as we do in T_S is neither weaker nor stronger.

Our model differs from AnoA's model in the batch queries, the adaptive corruption, the arbitrary sessions and the use of notions instead of anonymity functions. Instead of *batch queries* AnoA distinguishes between input, i.e. communications that are equal for both scenarios, and challenge queries, i.e. challenge rows. Input queries are always valid in AnoA. They are also valid in our model, because all the privacy aspects used for our notions equivalent to AnoA's hold true for identical batches without $\Diamond$ and $\Diamond$ is not allowed in the equivalent notions. In AnoA's single-message anonymity functions only a limited number of challenge queries, i.e. challenge rows, is allowed per challenge. We ensure this restriction with the restriction of using at most $\#cr$ challenge rows $CR_{\#cr}$. In AnoA, the adversary gets information after every communication. This is equivalent to multiple batches of size one in our case. We assume that for the analyzed protocol a protocol model can be created, which reveals the same or less information when it is invoked on a sequence of communications at once instead of being invoked for every single communication. Our notions, which match the AnoA notions, allow for batches of size one. So, our batch concept neither strengthens nor weakens the adversary.

AnoA's *corruption* is static, does not protect corrupted users[11] and AnoA includes restrictions on *sessions*. Hence, AnoA's notions translate to ours with the static corruption X_{c^-}, the corrupted communication have to be equal in both scenarios X_{c^e} and the session option of our model X_s.

[10]If $X_1 \implies X_2$ and $X_1 \not\Rightarrow X_3$, it follows that $X_2 \not\Rightarrow X_3$.

[11]Although AnoA does not explicitly state this, we understand the analysis and notions of AnoA this way, as scenarios differing in the messages corrupted users send/receive could be trivially distinguished.

AnoA's challenger does not only check properties, but modifies the batches with the *anonymity functions*. However, the modification results in one of at most four batches. We require those four batches (as combination of scenario and instances) as input from the adversary, because it is more intuitive that all possible scenarios stem from the adversary. This neither increases nor reduces the information the adversary learns, since she knows the challenger algorithm.

Table 3.5: Equivalences ($\langle X\rangle$ equivalence of X used, CR_i limits the number of challenge rows to at most i; k is the number of batches)

Framework	**Notion**	**Equivalent to**
AnoA	α_{SA}	$S\overline{O}_{c^e c^- s CR_1}$
	α_{RA}	$R\overline{O}[M\overline{O} - \lvert M\rvert]_{c^e c^- s CR_1}$
	α_{REL}	$(SR)\overline{O}_{c^e c^- s CR_2}$
	α_{UL}	$(2S)\overline{O}_{c^e c^- s CR_2}$
	α_{sSA}	$S\overline{O}_{c^e c^- s}$
	α_{sRA}	$R\overline{O}[M\overline{O} - \lvert M\rvert]_{c^e c^- s}$
	α_{sREL}[12]	$(SR)\overline{O}_{c^e c^- s}$
	α_{sUL}[13]	$(2S)\overline{O}_{c^e c^- s}$
Bohli's	$S/SA = R/SA$	$\overline{O}$
	R/SUP	$S\overline{O}\{R\overline{O} - \lvert U'\rvert\}$
	R/WUP	$S\overline{O}\{R\overline{O} - H'\}$
	R/PS	$S\overline{O}\{R\overline{O} - P'\}$
	R/SUU	$S\overline{O}\{RF\overline{L}\}$
	R/WUU	$S\overline{O}\{RF\overline{L} - H'\}$
	R/AN	$S\overline{O}\{RF\overline{L} - P'\}$
	R/WU	$S\overline{O}\{RM\overline{L}\}$
	R/WA	$S\overline{O}\{RM\overline{L} - P'\}$
	S/SA°	$S\overline{O}$
	S/SUP°	$S\overline{O} - \lvert U\rvert$
	S/WUP°	$S\overline{O} - H$
	S/PS°	$S\overline{O} - P$
	S/SUU°	$SF\overline{L}$
	S/WUU°	$SF\overline{L} - H$
	S/AN°	$SF\overline{L} - P$
	S/WU°	$SM\overline{L}$
	S/WA°	$SM\overline{L} - P$
	$S/X, R/X^\circ$	analogous
	X^+	$\langle X\rangle_{c^e}$
	X^*	$\langle X^\circ\rangle_{c^e}$
Hevia's	UO	$C\overline{O}_{c^0}$, $k = 1$
	SRA	$\overline{O}_{c^0}$, $k = 1$
	SA^*	$S\overline{O}\{RM\overline{L}\}_{c^0}$, $k = 1$
	SA	$S\overline{O}_{c^0}$, $k = 1$
	UL	$M\overline{O}[M\overline{L}]_{c^0}$, $k = 1$
	SUL	$SM\overline{L}_{c^0}$, $k = 1$
	RA^*, RUL, RA	analogous
Gelernter's	$R^{H,\tau}_{SA}$	$R^{H,\tau}_{SA\ c^0} \iff S\overline{O} - P_{c^0}$, $k = 1$
	$R^{H,\tau}_{SUL}$	$R^{H,\tau}_{S\overline{L}\ c^0} \iff SM\overline{L} - P_{c^0}$, $k = 1$
	R_X	analogous Hevia: $\langle X\rangle$
	R^H_X	analogous Hevia: $\langle X\rangle_{c^{\mathrm{sr}}}$
	$\hat{R}^H_X$	analogous Hevia $\langle X\rangle_{c^s}$

Bohli's Framework. Bohli and Pashalidis [26] built a hierarchy of application-independent privacy notions based on what they define as "interesting properties", that the adversary is or is not allowed to learn. Additionally, they compare their notions to Hevia's, which we introduce next, and find equivalences.

It is easy to see, that our definitions of U, Q, H (P is not easy and hence, explained more detailed below) match the ones of Bohli's properties (who sent, how often any sender sent and how many

[12]Under the assumption that in all cases m_0 is communicated like in α_{REL} of [11] and in α_{SREL} of one older AnoA version [10].

[13]Under the assumption that the receiver in Stage 2 can be another than in Stage 1 like in α_{UL} of [11].

senders sent how often) although we do not use a function that links every output message with the sender(/receiver), but the sender-messages-sets(/receiver-messages-sets). Bohli and Pashalidis additionally define the restriction of picking their communications equal except for the user (depending on the current notion sender or receiver) $\circ$. This is the same as allowing only the senders respectively receivers to differ (E_S resp. E_R).

Conceptually, our model differs from Bohli's model in the concept of challenges, the advantage definition, the order of outputs, and the allowed behavior of corrupted users.

Bohli's notions can be understood as one *challenge* ($n = 1$) with arbitrarily many challenge rows (any c). Further, it does not use a multiplicative term in its *advantage* ($\epsilon = 0$). Then δ equals the advantage, which has to be 0 to unconditionally provide a privacy notion or negligible to computationally provide this notion.

Bohli's framework assumes that the protocol outputs information as an information vector, where each entry belongs exactly to one communication. The adversary's goal in Bohli's framework is to link the index number of the output vector with the sender or receiver of the corresponding communication.

All except one of their properties can be determined given the batches of both scenarios. However, the linking relation property that partitions the index numbers of the output vector by user (sender or receiver depending on the notion), can only be calculated once the output order is known. Since our notions shall be independent from the analyzed protocol, the challenger cannot know the protocol and the way the output order is determined. Running the protocol on both scenarios might falsely result in differing output orders for non-deterministic protocols.

Thus, we adapt the linking relation for ACNs to be computable based on the batches. The interesting output elements the adversary tries to link in ACNs are messages. Hence, here the linking relation partitions the set of all messages into the sets of messages sent/received by the same user, which can be calculated based on the batches. This adaption is more restrictive for an adversary, since the partition of output numbers can be equal for both scenarios even though the sent messages are not. However, if the adversary is able to link the output number to the message, she can calculate our new linking relations based on Bohli's.

Further, Bohli's framework allows for notions, where the *behavior of corrupted users* differs in the two scenarios. This means privacy of corrupted users is provided, i.e. the adversary wins if she can observe the behavior of corrupted users. Those notions are the ones without the option X_{c^e}.

To match our batch query, Bohli's input queries, which include communications of both scenarios, have to be combined with a nextBatch query, which signals to hand all previous inputs to the protocol.

Hevia's Framework. Hevia and Micciancio [82] define scenarios based on message matrices. Those message matrices specify who sends which message to whom, but not in which order the communications take place. Notions restrict different communication properties like the number or set of sent/received messages per fixed user, or the number of total messages. Further, they construct a hierarchy of their notions and give optimal ACN protocol transformations that, when applied, lead from weaker to stronger notions.

Mapping of the properties follows mainly from Bohli's and the equivalences between Bohli and Hevia (including the one we correct in the following paragraph). Besides this, only Hevia's Unobservability (UO), where the matrices can be picked arbitrarily, is new. However, this corresponds to our $\aleph$ property, that always returns TRUE and allows any arbitrary scenarios.

Our model differs from Hevia's, since ours considers the order of communications and allows for adaptive attacks, as well as corruption.

Our game allows to consider the *order of communications*. Analyzing protocol models that ignore the order will lead to identical results. However, protocol models that consider the order do not achieve a notion – although they would in Hevia's framework, if an attack based on the order exists.[14]

Most of Hevia's notions are already shown to match Bohli's with only one batch ($k = 1$) and no corruption (X_{c^0}) [26]. However, we have to correct two mappings: in [26] Hevia's Strong Sender Anonymity (SA^*), which requires the number of messages a receiver receives to be the same in both scenarios was mistakenly matched to Bohli's Sender Weak Unlinkability (S/WU^+), in which every sender sends the same number of messages in both scenarios. The needed restriction is realized in Bohli's R/WU^+ instead. The proof is analogous to Lemma 4.3 in [26]. The same reasoning leads to Bohli's Sender Weak Unlinkability (S/WU^+) as the mapping for Hevia's Strong Receiver Anonymity (RA^*).

Gelernter's Framework. Gelernter and Herzberg [72] extend Hevia's framework to include corrupted participants. Additionally, they show that under this strong adversary an ACN protocol achieving the strongest notions exists. However, they prove that any ACN protocol with this strength has to be inefficient, i.e. the message overhead is at least linear in the number of honest senders. Further, they introduce relaxed privacy notions that can be efficiently achieved.

The notions of Gelernter's framework build on Hevia's and add corruption, which is covered in our corruption options. Only the relaxed notions $R^{H,\tau}_{SA}$ and $R^{H,\tau}_{SUL}$ are not solely a corruption restriction. We define new notions as $R^{H,\tau}_{SA}{=}\not\Diamond{\wedge}G$ and $R^{H,\tau}_{S\overline{L}}= \not\Diamond{\wedge}Q \wedge G$ that are equivalent to some of the already introduced notions to make the mapping to the Gelernter's notions obvious. They use a new property G, in which scenarios are only allowed to differ in the sender names.

Definition 12 (Property G)**.** *Let $\mathcal{U}$ be the set of all senders, $s_{b_i} = \{(u, \{m_1, \dots, m_h\}) \bigm| u$ send message $m_1, \dots, m_h$ in batch $i\}$ the sender-messages sets for scenario $b \in \{0, 1\}$. We say that G is met, iff a permutation perm on $\mathcal{U}$ exists such that for all $(u, M) \in s_{0_k} : (perm(u), M) \in s_{1_k}$.*

Note that Gelernter's relaxed notions (indistinguishability between permuted scenarios) is described by our property G, the need for the existence of such a permutation.

Theorem 3. *It holds that*

$$\begin{aligned}(c, \epsilon, \delta) - R^{H,\tau}_{SA} &\iff (c, \epsilon, \delta) - S\overline{O} - P,\\ (c, \epsilon, \delta) - R^{H,\tau}_{S\overline{L}} &\iff (c, \epsilon, \delta) - SM\overline{L} - P.\end{aligned}$$

Proof. [Proof sketch] Similar to Theorem 1. (See Appendix A.5.2 for details.) □

3.6.2 Use Case: Analyzing Loopix's Privacy Goals

To check if we include currently-used privacy goals, we decide on a current ACN that has defined its goals based on an existing analytical framework and which has already been analyzed: the Loopix anonymity system [128]. In this section, we show that the privacy goals of Loopix map to notions that we have defined (although the naming differs). Loopix aims for Sender-Receiver Third-Party Unlinkability, Sender online Unobservability and Receiver Unobservability.

Sender-Receiver Third-Party Unlinkability. Sender-Receiver Third-Party Unlinkability means that an adversary cannot distinguish scenarios where two receivers are switched:

> "The senders and receivers should be unlinkable by any unauthorized party. Thus, we consider an adversary that wants to infer whether two users are communicating. We define *sender-receiver third party unlinkability* as the inability of the adversary to distinguish whether $\{S_1 \to R_1, S_2 \to R_2\}$ or $\{S_1 \to R_2, S_2 \to R_1\}$ for any concurrently online honest senders S_1, S_2 and honest receivers R_1, R_2 of the adversary's choice." [128]

[14] Creating an adapted version left a degree of freedom. Our choice of adaptation corresponds with the interpretation of Hevia's framework that was used, but not made explicit in Bohli's framework.

Table 3.6: Definition of the Loopix notions

Notion	Name	Aspects
$LS\overline{O}$	Loopix's Sender Unobservavility	$E_\Diamond$
$LR\overline{O}$	Loopix's Receiver Unobservability	$E_\Diamond$
$S\overline{O}'$	Restricted Sender Unobservability	$\not\rightarrow \wedge E_S$
$R\overline{O}'$	Restricted Receiver Unobservability	$\not\rightarrow ' \wedge E_R$

The definition in Loopix allows the two scenarios to be distinguished by learning the first receiver. We interpret the notion such that it is only broken if the adversary learns a sender-receiver-pair, which we assume is what is meant in [128]. This means that the sender and receiver of a communication must be learned and is exactly the goal that motivated our introduction of complex properties: $(SR)\overline{L}$.

Unobservability. In Sender online Unobservability the adversary cannot distinguish whether an adversary-chosen sender communicates ($\{S \rightarrow\}$) or not ($\{S \not\rightarrow\}$):

> "Whether or not senders are communicating should be hidden from an unauthorized third party. We define *sender online unobservability* as the inability of an adversary to decide whether a specific sender S is communicating with any receiver $\{S \rightarrow\}$ or not $\{S \not\rightarrow\}$, for any concurrently online honest sender S of the adversary's choice." [128]

Receiver Unobservability is defined analogously.

Those definitions are open to interpretation. On the one hand, $\{S \not\rightarrow\}$ can mean that there is no corresponding communication in the other scenario. This corresponds to our $\Diamond$ and the definition of $LS\overline{O}$ and $LR\overline{O}$ according to Table 3.6. When a sender is not sending in one of the two scenarios, this means that there will be a receiver receiving in the other, but not in this scenario. Hence, $LS\overline{O}$ can be broken by learning about receivers and the two notions are equal. These notions are equivalent to $C\overline{O}$:

Theorem 4. *It holds that*

$$(c, \epsilon, \delta) - C\overline{O} \Rightarrow (c, \epsilon, \delta) - LS\overline{O}_{CR_1}.$$

$$(c, \epsilon, \delta) - C\overline{O} \Leftarrow (2c, \epsilon, \delta) - LS\overline{O}_{CR_1}.$$

Proof. [Proof sketch] Similar to Theorem 1. (See Appendix A.5.2 for details.) □

On the other hand, $\{S \not\rightarrow\}$ can mean that sender u does not send anything in this challenge. In this case, the receivers can experience the same behavior in both scenarios and the notions differ. We formulate these notions as $S\overline{O}'$ and $R\overline{O}'$ according to Table 3.6. Therefore, we need a new property that some sender/receiver is not participating in any communication in the second scenario:

Definition 13 (Property $\not\rightarrow$)**.** *Let u be the sender of the first scenario in the first challenge row of this challenge. We say that $\not\rightarrow$ is fulfilled iff for all j : $u_{1_j} \neq u$. (Property $\not\rightarrow '$ is defined analogously for receivers.)*

Theorem 5. *It holds that*

$$(c, \epsilon, \delta) - S\overline{O} \Rightarrow (c, \epsilon, \delta) - S\overline{O}' \text{ and}$$
$$(c, 0, 2\delta) - S\overline{O} \Leftarrow (c, 0, \delta) - S\overline{O}'.$$

Proof. [Proof sketch] Similar to Theorem 1. (See Appendix A.5.2 for details.) □

Remark. We do not claim that the Loopix system achieves or does not achieve any of these notions, since we based our analysis on the definitions of their goals, which were not sufficient to unambiguously derive the corresponding notions.

3.7. Climbing the Hierarchy: Towards ACN Primitives

Investigating state-of-the-art ACNs, we are interested in a first mapping of how they protect which property of the communication. Following the idea of Hevia and Micciancio [82] that discussed three fundamental concepts to invert all but one implication in their hierarchy of nine distinct privacy notions, we provide preliminary results on anonymization primitives for our much more extensive notion hierarchy.

3.7.1 Introducing Primitives

ACNs usually combine multiple techniques to provide privacy protection. However, as noticed in many surveys [45, 61, 133, 143], the approaches often share similar basic techniques, like (layered) encryption, indirection, shuffling/mixing, secret sharing and even dummy messages in a myriad of variations. Informally, we consider any such technique, i.e. any basic building block of which complete ACNs are composed of, as a *primitive*.

Example: *Broadcast. One primitive is a classical broadcast, where the sender sends her chosen message to all receivers. Thereby all information about the receivers, including the intended receiver of a message, the receiving frequencies of each user and also the sender-receiver relationship, is hidden.*

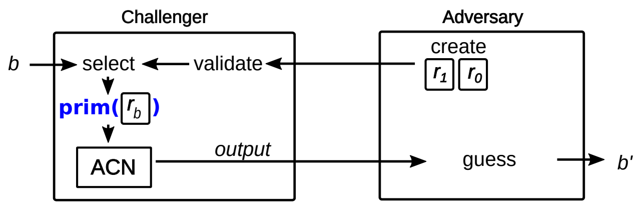


Figure 3.5: A primitive `prim` is a function applied by the challenger to the batches of the selected scenario.

For our game model (see Figure 3.5), we can understand any primitive as a function `prim`. The challenger applies it to the selected input batch after validating both input batches. From the perspective of an ACN, the primitive thus is a wrapper. The combination of the old ACN and the wrapper define the new ACN. Knowing the protection of the old ACN, we are interested in the protection of the new ACN. For two notions X_{strong}, X_{weak} with $X_{strong} \implies X_{weak}$: If any (old) protocol ACN achieves X_{weak} then we search for a primitive `prim` such that `prim`(ACN) achieves X_{strong}, or short:

$$X_{weak} \xRightarrow{\texttt{prim}} X_{strong}.$$

To have a way to express that no protection is required, we add the Null notion as the weakest possible notion to our hierarchy. It only accepts identical scenarios from the adversary and thus every protocol achieves it.

Example (cont.): *As a function the broadcast primitive* `Broadcast` *adds for each communication and user another communication with the same sender and message as the original one in a fixed receiver order (see Figure 3.6, blue part). Thereby any differences of the receivers in the batches are removed and Null* $\xRightarrow{\texttt{Broadcast}} R\overline{O}$.

3.7.2 Primitives Overview

Analyzing proposed networks with the goal to learn how to navigate through the hierarchy of protection goals, we identified anonymization concepts – our primitives – against the GPA (see Figure 3.7). We group primitives into three main categories:

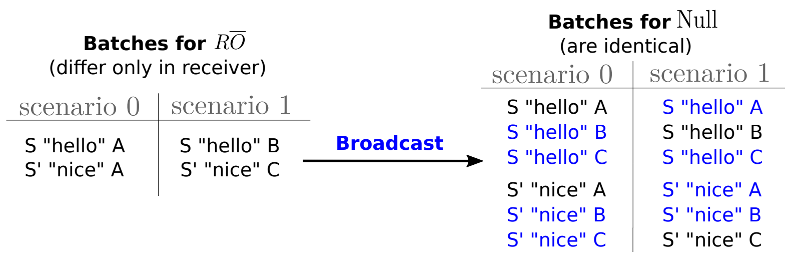


Figure 3.6: Example of the broadcast primitive for the senders S, S' and the receivers A, B, C. On input of the black communications, broadcast sends each communication to all receivers by adding the blue communications. If we assume that the attack on $\overline{RO}$ chooses the left batches and the protocol applies the broadcast primitive, then the adversarial observations correspond to the batches on the right, which of course are indistinguishable as they are identical.

1. *Encryption* to hide message content and its connection to users,
2. *Dummy Traffic* to hide the sending/receiving frequencies, activities and even user relationships, and
3. *Indirection and shuffling* to hide the connection between senders and receivers as well as the messages and timings.

A high-level overview of the primitives follows. We further argue their provided protection in Appendix A.8 and decompose four networks into primitives as examples in Appendix A.9. An in-depth formal model and analysis of the primitives including the detailed assumptions for practical realization, performance costs and composability investigations is, however, future work.

3.7.3 Encryption Primitives

We can either hide the message on the whole way trough the ACN via End-To-End Encryption or only on parts of their path in the ACN – until or from an intermediate node:

End-To-End Encryption (`E2EC`) To hide the content of the message in general, senders encrypt messages with the public key of the intended receiver.

Key private End-To-End Encryption (`E2EC(kp)`) To hide the same as `E2EC` *and* keep the protection for the receiver, `E2EC` is implemented with a key private asymmetric encryption scheme.

Encrypting on parts of the path (`EncToMix / EncFromMix`) To hide the linking between message content and sender (or receiver) it is sufficient to protect the message content on a part of the path. EncToMix uses an encrypted message from the sender until an intermediate party, EncFromMix from the intermediate party until the receiver.

3.7.4 Dummy Traffic Primitives

To confuse the adversary, protocols often add artificial traffic to the real communications. This *dummy traffic* does not transport any real messages from the sender to the recipient but is merely a means to hide different aspects of the actual desired (real) communications.

Overview. We intuitively know many ways to add dummy traffic. To hide the message length, we pad the message, i.e. add bits to reach a globally fixed length. To hide the frequency and activity

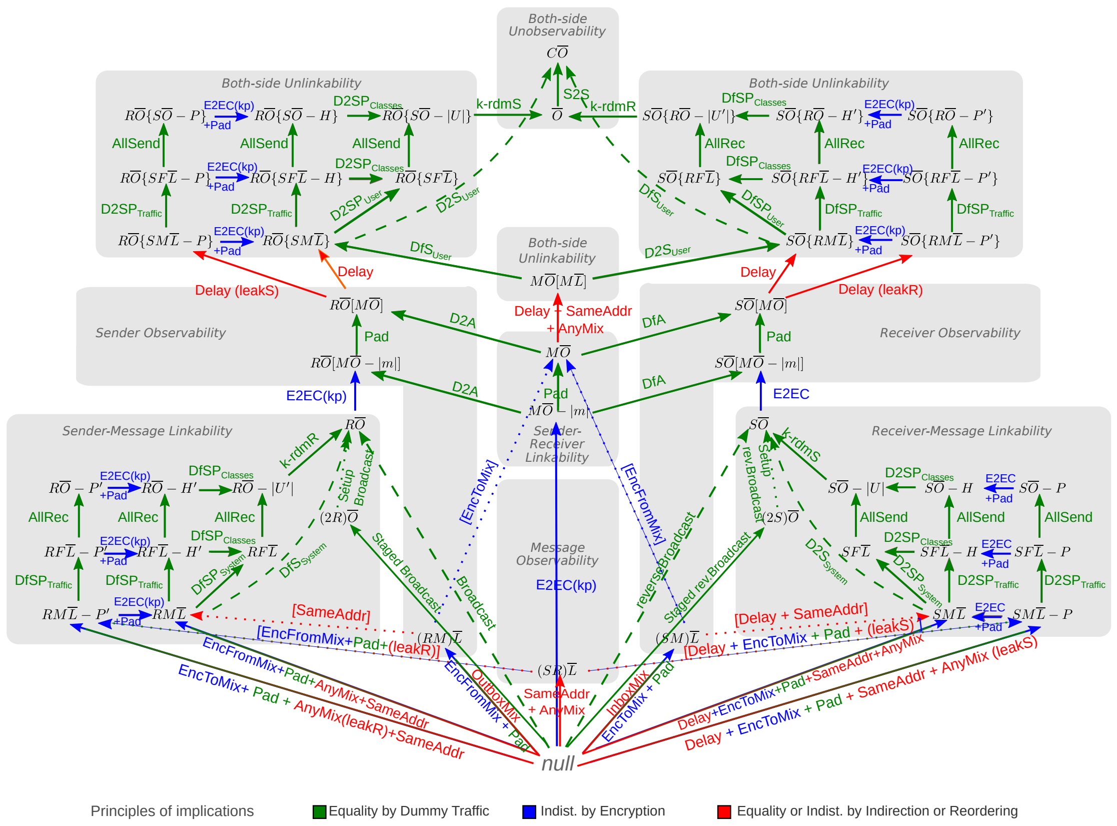


Figure 3.7: The figure shows the hierarchy of privacy notions from Section 3.5 but with inverted arrows[15], and the primitives used to invert the original implications from stronger notions at the top to weaker notions at the bottom (see Table 3.7 for an overview). The color indicates the high-level strategy that the primitive applies to hide the (meta) data. The *primitives in brackets* are not general for the implication, but only work given that the weaker notion protocol is achieved *using the stated primitives.*

patterns in the real behavior, we need to send more messages than before and thereby "pad" the number of communications to hide the attribute of interest, similarly to padding the messages to hide their real length. For such dummy traffic there is a variety of options (see Table 3.8): We can freely decide which dimension(s) of the communication (sending or receiving) are protected with additional "dummy" communications, and whether these use the real sender/message/receiver or a dummy instead. We might just decide on a subset of the real users (e.g. the ones being active in real communications this round) that generates/receives dummy traffic. Further, we might have different strategies to decide on how much padding we want to add. To hide whether or not someone is active, ensuring that the action is done once is enough. To protect frequencies, however, more padding is required.

Hiding the Message Length

Padding (pad) To hide the message length ($|m|$), all messages are padded to a fixed global length.

[15]We ignored $(SR)\overline{O}, (SM)\overline{O}, (RM)\overline{O}$ for this first investigation as they are likely reachable with either the primitive for $S\overline{O}, R\overline{O}, M\overline{O}$ or a combination thereof.

Table 3.7: Overview over Primitives (symmetric versions in brackets replace sending with receiving and vice versa)

Primitive	Effect
	Encryption Primitives
`E2EC` (`E2EC(kp)`)	the sender encrypts the message for the receiver
`EncToMix`/ `EncFromMix`	the sender encrypts the message for the mix / the mix encrypts the message for the receiver
	Primitives using dummy traffic
`pad`	all messages have the same length
`S2S`	total number of communications is fixed
`AllSend` (`AllRec`)	everybody sends at least once
$\mathtt{k-rdmS}$ ($\mathtt{k-rdmR}$)	as many senders as there are real messages this round sent at least once
$\mathtt{D2S}_{\mathtt{System}}$ ($\mathtt{DfS}_{\mathtt{System}}$)	everybody sends as many real + dummy messages as the system can deliver real messages
$\mathtt{D2S}_{\mathtt{User}}$ ($\mathtt{DfS}_{\mathtt{User}}$)	everybody sends a fixed amount of real+dummy messages ($<$ system maximum)
$\mathtt{D2S}_{\mathtt{Traffic}}$ ($\mathtt{DfS}_{\mathtt{Traffic}}$)	everybody sends as many real+dummy messages according as the user with the highest traffic amount this round
$\mathtt{D2S}_{\mathtt{Classes}}$ ($\mathtt{DfS}_{\mathtt{Classes}}$)	everybody sends a fixed amount of real+dummy messages according to her traffic amount
`D2SP` (`DfSP`)	only users that want to send something this round comply with the dummy traffic requirements
`D2A` (`DfA`)	everybody receives a (real or dummy) message whenever a real message is sent
`Broadcast` (`reverseBroadcast`)	everybody receives the real message
Staged (Setup)	the protective primitive is not (resp. only) applied during setup
	Indirection Primitives
`Delay`	the senders wait for a random delay before sending the message
`InboxMix` (`OutboxMix`)	messages are sent to a receiver-specific mix and delayed for a random amount of time each
`SameAddr`	replace the addresses of the real sender/receiver with a proxy; all messages are sent to the same proxy (independent of sender or receiver of the communication)
`leakS`(`leakR`)	which message belongs to which sender is leaked

Hiding the total number of communications

Source To Sink (`S2S`) To hide the number of real communications, dummy communications between two dedicated parties, the `Source` and the `Sink`, are added to pad the total number of communications to the fixed system maximum of real communications.

Hiding the User Behavior

We hide the sending activity and frequency of a user by padding the number of their sending actions. The variants differ in whether they choose a dummy communication partner (`Sink` as partner for real senders), a dummy message, or both, or if they use the original message and real users. Similarly, receiving activity and frequency is hidden by padding the number of receiving actions.

Table 3.8: Dummy Traffic Primitive Components

Dimension	Options
Message	real / dummy
Sender	real / dummy
Receiver	real / dummy
Apply for	all / participating users / random subset
Pad to	≥ 1, system maximum, per user maximum, per round max, dependent on user behavior

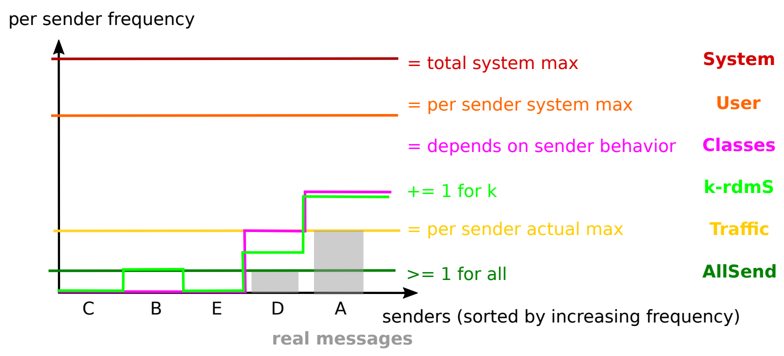


Figure 3.8: Padding options for *Dummy to Sink* illustrated

Dummy messages to dummy communication partner

Dummy to Sink (D2S) To hide aspects of the sending activity and frequency per user, D2S pads the number of communications per sender by adding dummy communications to a special receiver, the Sink. The actual effects of this padding depend on the chosen strategy of how many dummy communications to add per user. Ensuring, e.g., that everyone sends once, hides each sender's activity, but still leaks information about the sending frequency. We give an overview of the different strategies in Figure 3.8.

1. AllSend: To hide which senders are active (U), AllSend ensures that any sender sends *at least once* per round by adding dummy communications to the Sink. Note that the *number* of added dummy communications still leaks how many senders ($|U|$) were active before adding dummies.

2. $\mathtt{k-rdmS}$: Naturally, there cannot be more active senders than real communications. Thus, to hide the number of active senders ($|U|$), $\mathtt{k-rdmS}$ ensures that *as many senders send at least once per round as there are real communications* by randomly selecting inactive users for dummy communications. To ensure that the number of dummy communications is independent of the number of selected users, *all* senders (both active and randomly selected) send a dummy message to the Sink.

3. $\mathtt{D2S}_{\mathtt{System}}$ - To hide the sending frequency (Q) unconditionally, $\mathtt{D2S}_{\mathtt{System}}$ pads the sending events per sender to *the fixed system maximum of real messages per round.* This includes introducing new communication rounds if the system maximum would be violated otherwise.

4. $\mathtt{D2S}_{\mathtt{User}}$ - To hide the sending frequency (Q) given that the messages and real receiving events are not observable, $\mathtt{D2S}_{\mathtt{User}}$ pads to a fixed maximum of messages *per sender* and round. This includes introducing new rounds if the system maximum would be violated otherwise.

5. $\mathtt{D2S}_{\mathtt{Traffic}}$ - To hide the sending frequency (Q) given that the sending histogram (H) can be known, $\mathtt{D2S}_{\mathtt{Traffic}}$ pads to *the maximum of all real sending frequencies in this round.*

6. $\mathtt{D2S}_{\mathtt{Classes}}$ - To hide the sending histogram (H) given that the number of active senders ($|U|$) can be known, $\mathtt{D2S}_{\mathtt{Classes}}$ pads the sender of the most real messages to the greatest possible maximum of real messages a single sender sent in this batch, the sender of the second most real messages to this maximum minus one and so on. The maximum is determined based on the number of real messages and active senders per batch.

Dummy from Source (DfS) Similar to D2S, DfS hides aspects of receiving behavior, by padding the input of each receiver with dummy messages from a special sender, the Source. Note, however, that the corresponding trust assumptions are not fully symmetrical. Naive padding for receivers requires knowledge of the real traffic at the Source. Actual implementations of DfS, like Private Information Retrieval, can reduce the knowledge at the Source.

Dummy to Sink Participating/ Dummy from Source Participating (`D2SP/DfSP`) To hide the sending frequencies (Q) *only of active senders*, we apply the versions of `D2S` (except for `AllSend`, $\texttt{k} - \texttt{rdmS}$) only to the participating senders in each round. (Similarly for `DfS` and receivers.)

Dummy messages to real users

Dummy from All/ Dummy to All (`DfA/D2A`) To hide the sending frequencies (Q), the active senders (U) and the sender-receiver linking, whenever a sender sends a real message, `DfA` adds a dummy message from all other senders to the intended receiver of the real message. (Similarly, with `D2A` a sender sends the real message only to the intended receiver, but dummy messages to all other receivers.)

Note that `DfA` has a similar effect as `D2S` with the major differences that i) `DfA` is applied *per real message* and thus does not hide the number of real communications, but also does not need any additional party like the `Sink` and ii) by using users instead of the `Sink`, it hides sender-receiver relationships if the messages are indistinguishable.

Real messages to real users

(Reverse) Broadcast (`Broadcast/reverseBroadcast`) To hide any association between receivers and their receiving behavior, `Broadcast` lets each sender send each real message in a fixed order to all receivers. This order is independent of the intended receiver. (Similarly, `reverseBroadcast` hides the same information on the sender side, by letting every sender send all real messages of that round. Note, however, that the corresponding trust assumptions are not fully symmetrical. Naively for `reverseBroadcast` the (trusted) senders need a protected way to learn the message to be sent. Implementations like secret sharing in DC-nets without collisions however realize this primitive under realistically acceptable trust assumptions.)

Setup Protection

Staged Primitives Consider an ACN protocol, that protects established communications but provides no protection during the setup. We model this by “staging” the technique, i.e. the primitive is not applied in the first round of communication, but starts to be used in the second round.

Setup Primitives As counterpart to the staged primitives, we introduce the primitives with setup protection. These provide the hiding as the original primitive, but only for the setup phase.

3.7.5 Indirection Primitives

ACNs commonly employ additional steps in between the sender and receiver to unlink them. An intermediate node (proxy or mix) replaces sender and receiver addresses, as well as obfuscates the linking between incoming and outgoing traffic with the help of timing differences.

Typically unlinking only a single aspect (like time or addresses) is not enough, but only the protection of multiple such aspects assures the desired metadata protection. While each aspect on its own might seem artificial and in practice they are usually found only in combination, we first introduce primitives for any aspect and later discuss which combination hides which metadata.

Delay (`Delay`) To hide the original timing of the intention to send (and thereby the sending order), each sender picks a random time within the round for each of her messages.

(Inbox/Outbox) Mixing (`InboxMix/OutboxMix`) To hide the *timing* relation between sender and message (or receiver and message), we introduce a mixing postbox for each receiver (or sender). In `InboxMix`, all messages for a receiver are sent to the postbox of this receiver, an intermediate node specifically designated for that receiver, that shuffles them before sending them to the receiver. (`OutboxMix` similarly)

Sometimes we do not care whether we have an `InboxMix` or an `OutboxMix`. In these cases we use `AnyMix` to represent that it can be any of the two.

Address rewriting (`SameAddr`) To hide the original sender or receiver, the address of the receiver is replaced with and the message sent to the same proxy for the first part of *all* communications (until the proxy) and the address of the sender with the proxy for the last part (from the proxy). The proxy derives and addresses the final receiver.

Leak (`leakS/ leakR`) For some functionalities or performance, a protocol might use sender or receiver long-term pseudonyms. The above primitives can be weakened to a "leaky" version that allows the adversary to learn the pseudonyms and the pseudonym-communication linking (P/P'), but not the identity of the sender.

3.7.6 Effects in the Hierarchy

By putting the primitives into the hierarchy, we can observe patterns. Firstly, to achieve the very weak notions that merely unlink sender from receivers or messages, we need a broad combination of different techniques (encryption, indirection, padding) to ensure that we unlink all aspects of the communication. This is also the main area where indirection primitives find their application. Precisely, by using a proxy as receiver (`SameAddr`) and mixing (`InboxMix/OutboxMix`) we can get rid of the sender-receiver relation while still leaking the message content completely, and thereby also the sender-message and the receiver-message linking (i.e. only sender-receiver pairs that want to communicate *the same message* build the anonymity set, Null $\rightarrow (SR)\overline{L}$).

With the help of message padding (`pad`), encryption on the first part of the path (`EncToMix`) and the mixing per receiver (`OutboxMix`), we can hide the sender-message relation, but still leak the sender-receiver and receiver-message linking, Null $\rightarrow (SM)\overline{L}$ (similarly for the receiver-message linking).

If we apply a common proxy to the encryption and mixing by using `SameAddr`, we use one mix for all communications and thus hide the sender-receiver linking additionally, but still not the receiver-message linking. Delay is further required whenever the order of senders already allows to distinguish the scenarios (like in most of our sender notions). This allows us to realize $(SM)\overline{L} \rightarrow SM\overline{L}$.

To ensure that frequencies and activities are hidden, we need dummy traffic. However, with certain kinds of dummy traffic (e.g. broadcast) we even hide the sender-receiver linking.

Above we discussed encryption (`EncToMix`) to unlink communication partners. Further, end-to-end encryption (`E2EC(kp)`) is of course used to protect the message ($M\overline{O}$), and to reduce the leaked information about which messages are sent/received by the same user.

3.8. Discussion

In this section, we present the lessons learned while creating our framework.

Learning about privacy goals. The need for formal definitions is emphasized by the mapping of Loopix's privacy goals to notions as example that less formal alternatives leave room for interpretation. Further, a result like our hierarchy would be much harder to achieve without formal definitions.

These definitions allow us to point out the relation of privacy and confidentiality (Message Unobservability leaking Message Length $M\overline{O} - |M|$). The way we ordered the notions in the hierarchy allows easy identification of the notions implying $M\overline{O} - |M|$ (the middle of the upper part). Note that any privacy notion implying $M\overline{O} - |M|$ can be broken by distinguishing the message's content. Further, nearly all those notions also imply $M\overline{O}$ and hence, all such notions can be broken by learning the message length.

Our formal definitions also enabled the comparison of existing frameworks. Excluding differences in the adversarial model, quantifications and restrictions that do not apply to all ACNs, we observe that equivalent definitions are often defined independently by the authors of the analytical frameworks. For this reason, we included the notions of the other frameworks in our hierarchy in Figure A.1 of Appendix A.4. $\overline{O}$, $S\overline{O} - P$, $SM\overline{L} - P$, $R\overline{O}\{SM\overline{L}\}$ and $SM\overline{L}$ are defined (under different names) in multiple works; $S\overline{O}$ is even defined in all works.

Although previous work includes equivalent definitions, we realized that some notions are still missing. For example, we added weak notions like $(SM)\overline{L}$, $(RM)\overline{L}$ and $(SR)\overline{L}$ because they match our understanding of anonymity. Our understanding was confirmed by the analysis of Loopix' goals. Further, we defined all analogous notions for all communication parties involved (senders and receivers) as real-world application define which party is more vulnerable. For the concrete applications we refer the reader to Appendix A.6.

Consequently, we present a broad selection of privacy notions. We are aware that understanding them all in detail is a challenging task, so we want to provide intuitions and preferences, based on what we know and conjecture. We expect the lower part of the hierarchy to be more important for ACN*s* that do not trust the communication partner as [72] already includes an inefficiency result for $S\overline{O}$ and thus for all notions implying $S\overline{O}$. As a first guess, we think $S\overline{O}$ and $SF\overline{L}$, if higher overhead is manageable, $SM\overline{L}$, $(SM)\overline{L}$ (and receiver counterparts) and $(SR)\overline{L}$ are the most popular notions for ACNs without trust in the communication partner (similar for the corresponding receiver notions). With trust in the communication partner, we expect especially the strongest goals $\overline{O}$ and $C\overline{O}$ to be useful. Further, besides our discussion of primitives in Section 3.7, we want to add some results concerning two well-known systems to ease intuition. [11]'s analysis of Tor results in a small, but non-negligible probability to break $S\overline{O}$ and thus Tor does not achieve $S\overline{O}$ with our strict definition. Classical DC-Nets, on the other hand, do achieve at least $S\overline{O} - P$ [72]. To further more analysis results in the future, we provide quick instructions for a proof based on our notions in Appendix A.7.

Correcting Inconsistencies. While the above similarities most likely stem from the influence of prior informal work on privacy goals, attempts to provide concrete mappings have led to contradictions. The AnoA framework maps its notions to their interpretation of Pfitzmann and Hansen's terminology. Pfitzmann and Hansen match their terminology to the notions of Hevia's framework. This means that, notions of AnoA and Hevia's framework are indirectly mapped. However, those notions are not equivalent. While AnoA's Sender Anonymity and Hevia's Sender Unlinkability are both mapped to Pfitzmann and Hansen's Sender Anonymity, they differ: In Hevia's Sender Unlinkability the number of times every sender sends can leak to the adversary, but in AnoA's Sender Anonymity it cannot.

We believe that AnoA's Sender Anonymity should be called "Sender Unobservability", which is also our name for the corresponding notion. This follows the naming proposal of Pfitzmann and Hansen and their mapping to Hevia. It is also more suitable because AnoA's Sender Anonymity can be broken by learning whether a certain sender is active, i.e. sends a real message, in the system ($u \in U_b$). In order to achieve this notion, all senders have to be unobservable. To verify this, we looked at how the notions of AnoA have been used. For example in [41] the protocol model contains an environment that lets all senders send randomly. Hence, U_b is hidden by the use of this environment. We are convinced that the information that is allowed to be disclosed should instead be part of the notion and not modified by an environment. Only then are the notions able to represent what information is protected by the protocol.

Another lesson learned by comparing privacy notions is the power of names, because they introduce intuitions. The fact that Hevia's Strong Sender Anonymity is equivalent to Bohli's Receiver Weak Unlinkability seems counter-intuitive, since a sender notion is translated to a receiver notion. This might also be the reason for the incorrect mapping in [26]. However, Bohli's Receiver Weak Unlinkability is named this way because receivers are the "interesting" users, whose communication is restricted. It does not restrict senders in any way and hence should be, in most cases, easier to break according to some information about the sender. This is why we and Hevia have classified it as a sender notion. An analogous argument explains why Bohli's Receiver Weak Anonymity R/WA implies the restricted case of Bohli's Sender Strong Anonymity S/SA°.

Use and Limits. Because there is no restriction in the use of protocol queries, the only restriction to what can be analyzed is what is modeled in the protocol model. So, if the protocol model includes e.g. insider corruption and active behavior of the insider, like delaying or modifying of messages, those functionality can be used via protocol queries. The same applies to timing; if the protocol model specifies that it expects protocol queries telling it, that x seconds passed and the adversary gets meaningful answers after these protocol messages and only an empty answer after batch queries (because they are only processed after some time passed), attacks on timings can be analyzed. However, the challenge is to include this in the protocol model. This model also defines the exact meaning of a batch query, whether messages of one batch are sent at the same time or in a sequence without interruption and specifies whether a synchronous or asynchronous communication model is used.

Defining the protocol model with the strongest adversary imaginable and restricting it later on with adversary classes is a way to limit the work, when analyzing against different adversary models. We decided not to increase the complexity of the framework further by adding interfaces for dimensions of adversary models to the protocol model, i.e. adding more dedicated query types instead of the versatile protocol query. So far, our decisions for query types are driven by the related work and has been sufficient for our purposes. Future work will show whether refinements are necessary. Although we presented all notions that we deemed important, there might still be use cases that are not covered. With our properties as building blocks, we conjecture that it is easy to add the needed properties and use them in combination with ours. Further, for adding new notions to the hierarchy, our proofs can be used as templates.

4. Performance Limits of Anonymous Communication

Privacy always comes at a cost. In ACNs these costs are additional bandwidth and latency overhead. In this chapter we introduce, improve and compare known bounds on these costs for ACNs, as well as relate them to state-of-the-art protocols.

For this chapter, recall the basics from Section 2.1.1 and 2.1.5 and our privacy notions from Chapter 3. Results of this chapter have been published at WPES 2020 [99].

4.1. Bounds Overview

To increase the privacy of otherwise unprotected communication, ACN techniques necessarily create overhead. The dominating strategy to prove that a minimum amount of overhead is needed to achieve a privacy goal is based on attacks: According to assumptions and protocol requirements the attack is argued to succeed, unless the protocol introduces protective measures that have a minimum cost. We consider the protocol assumptions, privacy goal, adversary model, the attack idea and the derived performance bound as fundamental details of each bound.

Analyzing the proofs in the reports, we realized that their minimum amount of overhead is already necessary to achieve much weaker privacy goals for weaker adversaries than claimed in the works, and hence we tightened the bounds (and in one case correct the necessary overhead). Appendix B.1 describes this analysis in detail. Here we mention the improvements only briefly and then use the improved results throughout the rest of this chapter. Further, we focus on bounds for sender protection and refer the interested reader to Appendix B.3 for receiver focused bounds and pointers to other overhead considerations.

In this section, we give the extracts of the bounds in order of increasing privacy strength that they actually relate to. Their details are discussed as part of the comparison in the next section.

Dropping-Bound [6]

We call this bound "Dropping-Bound" because the bound describes the necessary overhead to prevent an attack that drops packets at the sender and recognizes the missing packets at the receiver.

Protocol Assumptions. The bound considers onion routing and mix networks. It relies on the implicit assumption that messages are successfully delivered with high probability.

Privacy Goal: $(SR)\overline{L}$. The paper [6] additionally proposes a protocol and thus only analyses for the strongest possible goal $C\overline{O}$. The bound, however, already applies for one of the weakest notions, $(SR)\overline{L}$. It defines that for any two (honest) senders the adversary must not learn which of them communicated with which of two receivers. Except this, she can learn anything, including e.g. how often each sender sends. Note that she can especially learn the fact that both candidate senders communicated with one of the two receivers, but not who communicated with whom.

Adversary Model. The paper states the assumption of active adversaries. Note, that the only necessary activity is to drop packets, though: The adversary can drop packets on the links of at least one sender and can observe at at least one receiver. Further, the adversary knows that this receiver expects a message[a].

Attack. The adversary chooses a candidate sender, drops as many messages sent by this sender as she can, and observes whether an expected message still arrives at the receiver, or not. She guesses her victim to be the real sender if no message arrives, and the alternative sender if it does.

Bound. Preventing this attack requires some overhead, which we can measure in added bandwidth and latency. Sending increasing numbers of redundant messages over alternative first hops requires higher bandwidth, but it improves the likelihood of delivery, as it reduces the chance that all paths start with adversarial links. Choosing longer paths increases latency but also the chance of an alternative message to be relayed through the victim sender and subsequently dropped by the adversary. This terminally reduces the accuracy of the adversary's guess[b]. The precise bound, which we discuss later, follows from calculating the adversary's advantage given an assumed cost.

[a]This is due to the formal definition of the privacy goal. Practically, we can however understand this as external information the adversary gained, e.g. because the application requires a stream of messages.

[b]Note that this assumes an integrated system model, in which users also act as intermediate nodes.

Trilemma [54]

The "Trilemma" bound claims that only two out of three desirable properties can be achieved in conjunction: low bandwidth overhead, low delays, and strong[a] anonymity. The Trilemma consists of bounds for different user behaviors, as well as adversary models. It describes the necessary overhead introduced by an attack that links packets based on timings and corruption of the intermediate nodes.

Protocol Assumptions. The analysis assumes only a single receiver, and two suspect senders. All messages are delivered in at most l_{max} rounds after sending.
Further, it considers the protocol to use a fixed amount of real and dummy messages per round. Two different user behaviors are specified: In the *synchronized model* one sender is assumed to send its real message and all other users synchronize to decide who sends dummy messages in this round. In the *unsynchronized model*, any sender sends their real message in the current round with the fixed probability p' and dummy messages with the fixed probability β.
Although no restriction in the type of protocol is made explicit, we expect the bound to hold only for onion routing and mix networks, as at least one intermediate node is assumed.

Privacy Goal: $(SM)\overline{L}$**.** While the report discusses $S\overline{O}$, the Trilemma already applies for one of the weakest notions $(SM)\overline{L}$. It defines that for any two (honest) senders the adversary cannot know which of them sent which message. Except this, the adversary can learn anything, including e.g. how often each sender sends. Note that she can especially learn the fact that both candidate senders sent a message, but not who sent which message.

Adversary Model. The Trilemma distinguishes two models:
The "non-compromising"[b] *adversary:* The attacker controls the receiver and the links adjacent to the two suspected senders.
The compromising adversary: The adversary additionally fully controls some intermediary nodes.

Attack. The paper discusses two ways of identifying the real sender upon reception of a message at the corrupted receiver.
The non-compromising adversary: First, the attacker monitors the sending behavior of both

suspected challenge users. If one user has not sent any message (real/dummy) within the l_{max} rounds before the considered message is received, the other must be the sender.
The compromising adversary: In addition to the attack above, the adversary follows a second strategy: With some probability she is able to observe all hops of either the challenge message, or the message sent by the alternative sender. She then can identify the sender-message pair and tell the sender of the challenge message.

Bound. Increasing either latency or bandwidth helps preventing these attacks: Sending dummy messages at higher probabilities translates to a larger set of candidate users that might have sent the message, and thus a better chance that the alternative suspect is in it. Increasing the number of hops, and hence the latency, reduces the chance of all intermediate nodes being corrupt, and also increases the interval during which the message may have been sent, which again translates to a larger set of candidate senders. The precise bound follows from calculating the probabilities of the above mentioned events in which the adversary can unambiguously identify the sender, subject to the assumed bandwidth and latency overhead.

[a] We show in App. B.1.4 that it also holds for a weaker definition of provable anonymity.
[b] This name is used to distinguish it from the compromising adversary, even though the non-compromising adversary compromises the receiver.

Counting-Bound [72]

The "Counting-Bound" relies on an attack that counts delivered packets and excludes candidates that did not sent enough packets.

Protocol Assumptions. None.

Privacy Goal: $S\bar{O}$**.** No information about any sender can leak. This includes for example that even if someone sent all messages, the adversary does not know whether or not she sent any message at all.

Adversary Model. The honest, but curious adversary corrupts all receivers and the links of at least one honest sender.

Attack. The Counting-Bound's privacy goal implies that all participating senders could have sent *all* real messages. The attacker now attempts to exclude at least one of them, by counting the number of messages they are sending. Knowing the number of real messages that are received (as the adversary controls the receivers), the adversary can exclude any sender who sent less messages.

Bound. The protocol cannot deliver more real messages to corrupt receivers than any sender sends in real and dummy messages.

Optimality-Bound [82]

We call this bound "Optimality-Bound" because it is included in Hevia and Micciancchio's proof that their way of adding dummy messages is optimal from a performance point of view. The idea for this bound is similar to the Counting-Bound.

Protocol Assumptions. All sent, real messages are delivered.

Privacy Goal: $S\bar{O}$**.** Alike the Counting-Bound.

Adversary Model. The adversary observes the links of at least one honest sender and knows how many real messages will be sent in total[a].

Attack. The adversary again tries to infer that some user did not send all real messages. Therefore, she counts the number of messages each sender sends and concludes that this sender cannot have sent all, if the number is less than the total amount of messages.

Bound. Each sender has to send as many (real and dummy) messages as real messages will be sent.

[a]This is due to the formal definition of the privacy goal. For practical reasons, we might however also think of this as external information the adversary gained through another channel.

4.2. Comparison

We first compare the Counting-Bound and Optimality-Bound, to find that they only differ in small nuances. After that we compare the remaining bounds, aspect by aspect.

4.2.1 Counting-Bound and Optimality-Bound are equivalent

Both bounds arise from the same argument: Considering a number of real messages that have been sent, anybody who sent less messages in total cannot have sent them all.

Protecting the privacy hence requires generating enough dummy messages to ensure that every sender sends as many times as real messages are delivered by the protocol. Indeed they use the same definition for the privacy goal, i.e. protected communication properties, their advantage definitions are equivalent (see Appendix B.2.1) and the bounds state the same overhead (see Appendix B.2.2).

While the privacy goal and resulting bound are identical, the authors of the two bounds looked at this from slightly different angles: The Counting-Bound does not have any assumptions on the protocol, but instead requires that the receiver is corrupted, such that the adversary can count the delivered messages. The Optimality-Bound however does not corrupt the receiver, but instead silently assumes that all messages are delivered and exploits the fact that the adversary knows how many real messages are sent in total. Therefore, the adversary trivially learns the number of delivered messages. Both derive the same bound, but their conclusions differ correspondingly: The Counting-Bound limits the number of delivered messages, while the Optimality-Bound requires the senders to send enough dummy messages.

We continue to use the Counting-Bound as representative for both.

4.2.2 Protocol Assumptions

The papers state, but also silently make assumptions regarding sending behavior, delivery guarantees, and supported protocols.

Sending Behavior. The Counting-Bound and Dropping-Bound[1] make no assumption about the distribution of sending events per round. The Trilemma however considers a specific sending behavior with fixed amounts of dummy and real messages per round, and their synchronized and unsynchronized sending model.

Delivery Guarantees. The Counting-Bound does not consider a maximum delivery delay. As only $\frac{1}{n}$ of the sent messages (dummy and real) reach their destination, some messages might not be delivered.

[1]This is not to be confused with the assumptions for the protocol proposed in the same paper [6], where the messages are sent at the same point in time.

The Dropping-Bound silently assumes successful message delivery. Missing messages otherwise could not be interpreted as successful attacks by the adversary, but they could be an artifact of the protocol.

The Trilemma assumes a maximum delay that the network adds. Note that in the synchronized setting this guarantees that all messages can successfully be sent and received, as users get assigned one of n rounds to send their message into the network. The unsynchronized setting in contrast does not provide this kind of certainty, since a user can only send her message based on the result of a coin flip. Therefore, in every round there is some probability that a certain user has not been able to send their message yet (even though this probability is negligible after enough rounds).

Protocol types. The Counting-Bound applies to all types of ACN protocols, as it only considers the number of sent and received messages. The Dropping-Bound, in contrast, only applies to onion routing and mix networks.

The Trilemma states that no protocol with a minimal latency of $l_{max} = 1$ can achieve their privacy goal, as the resulting advantages of their attack are non-negligible[2]. However, there exist protocols with this minimal latency achieving even stronger privacy notions against the considered adversary model, like the secure multi-party computation protocol as discussed in [72] or the well-known DC-Net, which is proven to achieve a stronger notion than targeted by the Trilemma in [72]. The authors recognize this limitation in later work [55], and we suspect their bound to apply only to ACNs following the onion routing or mix network paradigms.

4.2.3 Privacy Goals

Although all bounds claim to hold for the privacy goal "anonymity", the protection at which their overhead becomes necessary differs.

Batch Assumption. In our privacy game from Chapter 3, a batch is so far understood as a sequence of communications, but the semantics of a batch are not defined further before. For this comparison, we understand a batch as communications that start in an unpredictable order, at least for the adversary. The easiest way is to think of them being initiated simultaneously or in a random order. Formally, this requires using a random permutation over all communications of the batch.

The Counting-Bound targets $S\overline{O}$, the strongest goal of these analyses[3]. It is a very strong notion that protects not only the linking of sender-message and sender-receiver pairs, but even the frequency of sending, which for instance after a critical event could jeopardize the sender's safety.

Both other bounds target weaker notions with no direct relation to each other. The Trilemma considers $(SM)\overline{L}$, which only prevents linking sender-message pairs, while the notion of the Dropping-Bound only prevents linking sender-receiver pairs ($(SR)\overline{L}$). Both allow the adversary to succeed in linking other properties, and allow to learn, for example, the number of real messages each user has sent.

To visualize the extent of the difference, note that we actually define multiple other privacy goals in between the ones targeted by the different bounds in Chapter 3 (see Figure 4.1).

4.2.4 Advantage Definitions

The advantage definitions of the papers are equivalent. We show equivalence mostly with simple transformations. Only the Dropping-Bound represents a slight exception, but also its chosen

[2] $\delta \geq \frac{1}{2}$ for the unsynchronized and $\delta \geq 1$ for the synchronized setting (see Appendix B.2.5)

[3] It is called "sender anonymity" in [82] and is shown to map to $S\overline{O}$ in Section 3.6.1.

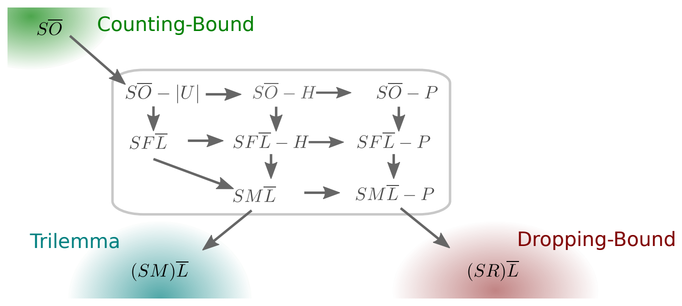


Figure 4.1: Excerpt of the hierarchy from Figure 3.3 of Section 3.5 with the privacy goals of the bounds highlighted. Arrows point from stronger to strictly weaker goals.

total variation distance can be shown equivalent using known results [50] (see Appendix B.2.1 for details).

4.2.5 Additional Restrictions

Corrupted users. The Dropping-Bound introduces the additional restriction that corrupted users send and receive the same messages in both scenarios, as the scenarios would otherwise be trivially distinguishable. Formally, this matches the corruption restriction and leads to $(SR)\overline{L}_{c^e}$. For our comparison we can however simply add this restriction to all notions. For the other bounds it does not change anything as it is already always fulfilled: all senders are assumed to be honest and the receivers have to receive the same messages per notion definition.

Allowed number of challenge rows. The Trilemma[4] and Dropping-Bound need only two differing communications (the ones whose senders are switched) in the compared scenarios. Thus, the notions of the Trilemma and Dropping-Bound are also in this regard weaker than the one of the Counting-Bound where multiple (precisely μ_{max}) differing communications are needed for the attack.

Further, the protocol model of the Trilemma allows only one user to send a real message per round and this permutation over the users is assumed to be chosen randomly. This fits our understanding of batches: The order, in which the chosen communications are input to the protocol model, is random.

4.2.6 Adversary Models

The bounds rely on different adversary models, which we depict in Figure 4.2 (the comparison to the models as stated in the papers is provided in Appendix B.1.1).

It is sufficient for all the attacks to compromise or influence outgoing links or the attached relays at the sender, as well as the receiver, as this enables to correlate events at the terminals of the communication. The adversaries in the *Counting-Bound* and *Trilemma in the non-compromising case* are virtually identical: one that corrupts all links of the victim sender(s) and the receiver. Their only difference lies in the number of victims, as the Trilemma considers two, and the Counting-Bound only a single victim to be monitored. The *Trilemma in the compromising case* additionally allows to passively compromise some intermediary protocol parties, i.e. to learn their keys and eavesdrop at them.

Only the *Dropping-Bound* allows the adversary to drop messages, and hence considers an active

[4]Even though the Trilemma [54]'s privacy notion formally allows just one communication to differ, its attack is, in combination with the assumption that every user sends exactly one message, not possible with only one differing communication.

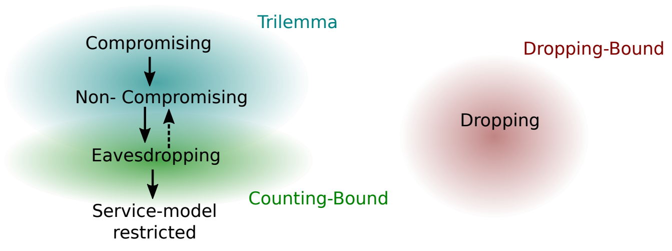


Figure 4.2: Hierarchy of adversary models. Hierarchically lower adversary model are weaker. The dotted arrow represents the additional relation caused by ignoring the number of observed victims.

model. Eavesdropping capabilities on the sender links are not strictly required, and it is at least conceivable that a remote adversary could cause the required message loss, for example by causing congestion on targeted links. Albeit this model is stronger than those of the other bounds in terms of behavior (active), it can as well be considered weaker in terms of the needed eavesdropping capabilities.

Note regarding the system models. Recall that in the service model, the receiver is not an active part of the network but an external entity (like in Tor). Corrupting a receiver in this case can be achieved in different ways. Beyond controlling the receiver herself, for unencrypted traffic it suffices to control only her network links (trivial for the ISP), or the last node on her anonymization path. Note, that for the Counting-Bound the traffic to the receiver can even be encrypted as only the fact that those are real messages is important.

We include the adversary model that only eavesdrops on these links as *service model restricted* to our comparison. It is weaker than those of the Counting-Bound and Trilemma.

4.2.7 Bounds

We explain the minimun cost as inferred by the proofs of the bounds in the following and compare them with each other. Therefore, we unify the notation of the different bounds as follows.

An ACN has n users. The set $\mathcal{U}$ includes all senders, $\mathcal{U}_H$ the h honest senders. If the privacy goal challenges the adversary to decide between two suspect senders, we call these "challenge senders u_0 and u_1" and u_1 the "alternative user to u_0". Further, we refer to the message of this communication as "critical" or "challenge message". λ is the security parameter and δ the advantage of the adversary in identifying the real sender.

The Trilemma requires a message to be delivered after at most l_{max} rounds/hops. We additionally write l_{exp} for the average of the number of hops.

Further, the Trilemma assumes dummy and real messages to be distributed uniformly over several rounds. We use β to denote the probability that a node sends a dummy message in a given round, p' for the probability of sending a real message. $p = p' + \beta$ is the total probability that a node sends in a round.

c_p is the number of intermediate nodes the adversary compromised passively (c_a for the number of actively compromised nodes).

Tables B.1 and B.2 of Appendix B show the connection to the original notation and summarize our notation.

Counting-Bound

Recall the idea from Section 4.1: To prevent that the adversary can exclude any sender as suspect for *all* sent messages, any sender has to sent at least that often. The total number of sending events for all senders $Com(r)$ has thus to at least cover that any sender sends $Out(r)$ messages:

$$Com(r) \geq Out(r) \cdot |\mathcal{U}_H| = Out(r) \cdot h$$

The bound shows a required overhead of at least $h-1$ dummy messages per message that reaches the destination. In other words, $\frac{h-1}{h}$ of messages are overhead because there are at least h times more sending events than received messages. The distribution of overhead during each round is flexible, as long as the sum of the overhead compensates for all delivered messages up to any specific round.

Note that the bound in [72] is given in the number of honest senders instead of all senders. As the Counting-Bound's adversary model does not include corrupted senders, both numbers are equivalent ($h = n$).

Trilemma

The Trilemma states the trade-offs for two adversaries (non-compromising and compromising), and two sending behaviors (synchronized and unsynchronized) and infers areas, i.e. bandwidth-latency combinations, where their privacy goal cannot be achieved.

Non-compromising adversary with synchronized users. Recall the attack: The adversary knows the two users out of which one has sent the challenge message. She also knows the interval, during which the message must have been sent. If she does not observe the alternative user sending a message in this interval, she knows the sender with certainty. Otherwise, she randomly accuses one of her two suspects.

Her advantage over guessing randomly equals the probability that the alternative user has not sent a message within the critical interval. In the synchronized setting, the probability that the alternative user *sent a message* in the critical time interval of l_{max} rounds, is bounded by the sum of the probabilities that she sent a real message[5]$(= \frac{l_{max}-2}{n-1})$ and the probability that she sent a dummy message[6] $(\leq \frac{\beta n(l_{max}-1)}{n-1})$. So the bound is simply the probability of the complementary event:

$$\begin{aligned}\delta \geq & 1 - min\left(1, \frac{(l_{max}-2) + \beta n(l_{max}-1)}{n-1}\right)\\ \geq & 1 - min\left(1, \frac{(l_{max}-1)(1+\beta n)}{n-1}\right)\end{aligned}$$

Comparison to Counting-Bound. This is intertwined with the considered privacy goals and protocol assumptions. The Counting-Bound aims at achieving $S\overline{O}$. To hide the sending frequency for the adversary model[7], everybody else has to send a dummy message, whenever a single user is sending a real message. The Trilemma aims at $(SM)\overline{L}$, which allows the number of dummy messages to be reduced, as the sending frequencies do not need to be hidden. Sending a real message as alternative sender in the critical time interval is enough to hide the sender-message linking, as desired. This joint sending is reflected in the first part of the sum $(\frac{l_{max}-1}{n-1})$ in the Trilemma for synchronized users, which therefore cannot be found in the Counting-Bound.

5 $\frac{\text{number of rounds (except sending of critical message)}}{\text{number of users (except real sender)}}$

6 $\frac{\text{number of dummy messages in the rounds}}{\text{number of users}}$

7 The Trilemma non-compromising and Counting-Bound's adversary model are the same except for the number of victims.

When we would require the latency to be minimal for their protocol model ($l_{max} = 2$), we force the Trilemma to only consider dummy messages of the current round as suitable cover: As per the protocol assumption in synchronized sending only one real message per round is sent and as the critical interval is just this round, only these dummy messages contribute to the hiding, just as for hiding frequencies in the Counting-Bound. We also see this reflected in the more precise formula by setting $l_{max} = 2$:

$$\delta \geq 1 - min\left(1, \frac{\beta n}{n-1}\right)$$

This advantage is 0 if $\beta = \frac{n-1}{n} = \frac{h-1}{h}$; exactly the overhead required in the Counting-Bound.

Non-compromising adversary with unsynchronized users. We know that an alternative user does not send in a specific round with probability $1-p$. Additionally, the choice of sending in the $l_{max} - 1$ rounds is independent. We hence can bound the probability that a second observed user does not send by

$$\delta \geq (1-p)^{l_{max}-1}$$

(cf. Appendix B.1.4 for a discussion of the original bound in this case).

Comparison to Counting-Bound. The Counting-Bound does not require the overhead to be evenly distributed over the rounds. We however temporarily assume so to allow for a comparison. Thereby the Counting-Bound requires the probability of a user sending any (real or dummy) message to be $p = p' + \beta = 1$. This resembles the improved Trilemma bound to minimal latency and the additional effects of a higher latency can be seen in the Trilemma, but not mapped to the Counting-Bound; as in the case with synchronized users.

Non-compromising adversary's area of impossibility. Based on the above bounds on the advantage, we can infer that for some combinations of latency l_{max} and bandwidth overhead β the considered attack has non-negligible advantage, i.e. the privacy goal is broken. Such parameter combinations constitute the *area of impossibility*.

If we e.g. assume that the message should be delivered after only one intermediate relay processed it, the adversary wins unless the alternative user sends in the same round (to this relay). Thus unless every user sends in every round (β approaches 1), the goal cannot be achieved for this short latency ($l_{max} = 2$). For the other extreme case of no dummy messages ($\beta = 0$), the adversary wins unless the alternative user sends her real message while the challenge message is routed. Thus unless the latency is very high (e.g. $l_{max} = n + 1$) and all users send their own message in the meantime with overwhelming probability, the privacy goal cannot be achieved.

The Trilemma makes the assumption that $n \approx poly(\lambda)$ and derives the following equations for the synchronized setting. All parameter combinations that fulfill them cannot achieve the privacy goal.

$$2(l_{max} - 1)\beta \leq 1 - \frac{1}{poly(\lambda)}, \beta n \geq 1$$

For the unsynchronized setting the equations are equal, except that β is replaced with p.

Comparison to Counting-Bound. The Counting-Bound and the area of impossibility from the Trilemma can be transformed to the following statements (cf. Appendix B.2.4):

$$\text{Counting-Bound: } \beta \geq \frac{Out(r)}{r}\left(1 - \frac{1}{poly(\lambda)}\right), p = 1$$

$$\text{Trilemma: } \beta \geq \frac{1}{2(l_{max}-1)}\left(1 - \frac{1}{poly(\lambda)}\right), \beta n \geq 1$$

$\frac{Out(r)}{r}$ is the average number of messages delivered to the destination in each round. We hence assume this number not to be much lower than 1 for most protocols, to retain utility.[8] Also, Section 4.2.2 yields that the Trilemma requires $l_{max} > 1$. In consequence, it holds that $\frac{1}{2(l_{max}-1)} < \frac{1}{2}$. This shows that the lower bound on the bandwidth overhead of the Counting-Bound is higher, reflecting its stricter requirements.

Compromising adversary. Extending the adversary to compromise up to $c_p \leq n-2$ intermediate nodes facilitates the attack of tracing messages along their anonymization paths, if all nodes on these paths are under adversarial control. This increases the advantage of the adversary, and the Trilemma is interested in this additional probability for an attack to succeed. We explain how the probability is bounded in the Appendix B.1.5 and only discuss the *area of impossibility for the compromising adversary* here.

If an adversary passively compromises $c_p < l_{max} - 1$ protocol parties, then the area of impossibility is

$$2(l_{max} - 1 - c_p)\beta \leq 1 - \frac{1}{poly(\lambda)}.$$

If the number of compromised parties is $c_p \geq l_{max} - 1$, then anonymity cannot be reached for

$$2(l_{max} - 1)\beta \leq 1 - \frac{1}{poly(\lambda)} \text{ and } l_{max} \in O(1).$$

Comparison to Counting-Bound. We have already compared the case without compromised protocol parties. With added compromised parties the result cannot directly be matched to the Counting-Bound (as it uses no compromised protocol parties). We can however again transform the impossibility area for the case of $c_p < l_{max} - 1$ to

$$\beta \geq \frac{1}{2(l_{max} - 1 - c_p)}\left(1 - \frac{1}{poly(\lambda)}\right)$$

Note, that this requires more bandwidth overhead than without compromised intermediate nodes, as expected. It is however still a weaker bandwidth requirement than in the Counting-Bound, as $\frac{1}{2(l_{max}-1-c_p)} < \frac{1}{2}$ and the Counting-Bound's factor $\left(\frac{Out(r)}{r}\right)$ is assumed to be close to 1. For the case of more corrupted parties, interestingly a constant latency is no longer possible as this ensures a non-negligible advantage that cannot be balanced with bandwidth.

Dropping-Bound

Recall the idea for the Dropping-Bound: The adversary has two suspects, from which one is sending to the receiver, in which the adversary is interested in. In her attack, she correlates her dropping of packets sent from the victim with the missing arrival of an expected packet at the receiver. This attack is successful unless one of two cases happens: 1) the communication of the alternative suspect is routed over the victim and thus the adversary wrongly accuses the victim even if the alternative suspect sends to the receiver or 2) the adversary cannot drop all copies of the message sent by the victim and thus wrongly acquits the victim.

This bound measures *onion cost* of a user as the expected number of packets (own and relayed messages) a user sends. The bound is based on two key observations: 1) if the victim forwards less packets than users exist (uses sublinear onion cost in the number of users), there is some user whose packet she does not forward, and 2) if the victim sends only few copies (logarithmic in the security parameter[9]) and the amount of corrupted nodes is high enough,[10] the adversary can drop

[8] In the Trilemma [54] the number of delivered messages per round $\frac{n}{n+(l_{max}-1)}$ approaches 1 for high numbers of users, for the protocol used in the Counting-Bound the number is 1.

[9] They implicitly assume that this is also sublinear in the number of users.

[10] They do not mention any requirement on the number of corrupted nodes except that the amount is constant, but if there are less corrupted nodes than copies of the message, she cannot succeed by dropping messages coming directly from the victim.

all copies with non-negligible probability. Combining the two observations, the attack leads to a non-negligible advantage. Thus, for their privacy goal, the onion cost per user has to increase faster than $\log \lambda$, i.e. be in $\omega(\log \lambda)$, and therefore the onion cost for the whole network has to be in $\omega(n \log \lambda)$.

Comparison. The authors of the Dropping-Bound assign the Trilemma an onion cost of $\omega(n)$, while their own result entails $\omega(n \cdot \log \lambda)$ due to the "stronger"[11] active adversary [6].

We cannot confirm this. The number of onions that are directly sent, i.e. the onion cost, is the number of onions created by all users multiplied by the rounds they stay in the network. To be able to compare the onion costs, we use $l_{exp} \leq l_{max}$ as the average latency of the protocol. The number of onions is the number of real messages plus the number of dummy messages. Both are multiplied with the number of rounds messages spend in the network (l_{exp}). Recall for the Trilemma: $\beta + p' = p$ messages per user and round are sent. There are n users as well as rounds – as every user is assumed to send one message, and only one user sends a real message per round. This results in $n^2 \cdot p$ messages in total that stay for l_{exp} hops in the network. So the onion cost per user in the Trilemma is $\frac{n^2 \cdot p \cdot l_{exp}}{n} = n \cdot p \cdot l_{exp}$, or $n^2 \cdot p \cdot l_{exp}$ for the complete network, which is considerably higher than a bound of $\omega(n)$ or $\omega(n \log \lambda)$.

Considering the impossibility area, the Trilemma states that $(SM)\overline{L}$ is impossible for onion cost of $\frac{n^2(poly(\lambda)-1)}{2poly(\lambda)} \approx n^2$.

The onion cost for the *Counting-Bound* for the whole network can be transformed to $\left(n^2(\frac{n-1}{n} + \frac{1}{n})\right) \cdot l_{exp} = n^2 \cdot l_{exp}$.

Note that also the precise onion cost for Counting-Bound is higher than for the Trilemma (since $p \leq 1$, $n^2 \cdot l_{exp} \geq n^2 \cdot p \cdot l_{exp}$). Both onion costs are higher than the one for the Dropping-Bound, contrary to the claim in [6].

Note on the Dropping-Bound in latency and bandwidth overhead. On the other hand, we can translate the onion cost of the Dropping-Bound into a bound on latency and bandwidth overhead under the assumption that every message stays in the network for the allowed latency. The resulting impossibility area confirms the above order in costs:

$$p \cdot n \cdot l_{max} > \log \lambda \iff p > \frac{\log \lambda}{poly\lambda \cdot l_{max}}$$

Intermediate Summary on the Overhead Comparison

All papers discuss the influence of bandwidth overhead on anonymity. The latency overhead is explicitly considered in the Trilemma and implicitly in the Dropping-Bound. To permit a comparison between the bounds, we transformed all bounds to account for the differing models and assumptions. In result, the overhead required by the Counting-Bound and Optimality-Bound is the highest, by the Trilemma the second highest and the Dropping-Bound, albeit based on an active adversary, introduces the lowest overhead.

4.2.8 Comparison across the Aspects

Interestingly, even though all bounds claim that strong anonymity comes with high overhead, their privacy goal, attacker model, protocol assumptions and also postulated cost differ considerably (cf. Table 4.1).

[11] It is not stronger in all dimensions as discussed in Section 4.2.6.

Table 4.1: Final Bounds Summary

Paper	**Notion**	**Adversary**	**Protocol Assumptions**	**Attack**	**Bound (comparable case, formal)**
[72]	$S\overline{O}$	eavesdrop + receiver	no restriction	count messages sent from victim; if more received they are not all from the victim	$\beta \geq \frac{Out(r)}{r}\left(1 - \frac{1}{poly(\lambda)}\right)$, $p = 1$
[82]	$S\overline{O}$	eavesdrop	guaranteed delivery for up to μ_{max} messages	count messages sent from victim; if more received they are not all from the victim	$\beta \geq \frac{Out(r)}{r}\left(1 - \frac{1}{poly(\lambda)}\right)$, $p = 1$
[54]	$(SM)\overline{L}$	eavesdrop + receiver + relay	required message delivery after l_{max} rounds; onion routing, mix nets, not applicable for DC-Nets	exclude senders that did not send in the time where the critical message was sent, if all relays corrupt: trace message	$\beta \geq \frac{1}{2(l_{max}-1)}\left(1 - \frac{1}{poly(\lambda)}\right)$, $\beta n \geq 1$
[6]	$(SR)\overline{L}$	active + receiver	delivery guaranteed (unless aborted), only onion routing, mixing	drop all messages send from the victim observe missing of expected message at receiver	$p > \frac{\log \lambda}{poly\lambda \cdot l_{max}}$

For the Optimality-Bound and the Counting-Bound we realized that cost and privacy goal are equal, and that the attacker models differ only slightly. The differences are easily explained by deviating protocol assumptions. Resulting, the Optimality-Bound and Counting-Bound have the highest cost and discuss the strongest privacy goal, albeit in face of a comparably weak adversary model, and without (Counting-Bound) or with minor (Optimality-Bound) restrictions on the protocol types. Thus, protecting $S\overline{O}$, which explicitly hides which sender sends how often, against an adversary that both observes the first link of the sender (e.g. her ISP) and the corresponding receiver (or has external knowledge about the number of received real messages) is indeed only possible with high bandwidth overhead.

The papers introducing the other two bounds state stronger, yet analyze weaker privacy goals, and postulate lower cost. The Trilemma aims at unlinking the sender from her message ($(SM)\overline{L}$), while the Dropping-Bound aims at unlinking pairs of senders and their receivers ($(SR)\overline{L}$). Note that for $S\overline{O}$ every sender sends a dummy message per real message to assure that real sending frequencies are hidden. For $(SM)\overline{L}$ (or $(SR)\overline{L}$) the bandwidth can be lower because it allows to learn that someone is a more active sender. $(SM)\overline{L}$ (or $(SR)\overline{L}$) only ensures that no one can link a certain message (or receiver) to the sender. On the other hand, compared to the Optimality-Bound and Counting-Bound, the adversary model in the Trilemma is slightly stronger. Interestingly, reducing the privacy notion, but using a slightly stronger adversary model for the Trilemma (and an incompatible adversary model for the Dropping-Bound), allows the bounds on the overhead to drop considerably. So, in this case the change in the adversary model cannot outbalance the change in the privacy goal.

The cost for unlinking sender and message in the Trilemma is higher than unlinking sender and receiver in the Dropping-Bound although the latter assumes an active adversary. The reasons are that the Trilemma is tailored to the special case that we needed to use to make the bounds comparable and that timing observations are exploited in the Trilemma. Further, the adversary in the Trilemma is not strictly weaker than the one for the Dropping-Bound. They indeed are incompatible, as the latter is stronger with respect to its behavior being active, whereas the former has a larger area of control, as it can compromise more and different parts of the network.

4.3. Implications

We extend the idea from [54] to contextualize our results with existing ACNs. The comparison to actual ACN protocols of course has to be taken with grains of salt: Exceeding the theoretical bounds in overhead indicates that an ACN *may*, but not that it actually *does* achieve the corresponding privacy notion. We discuss system classes, loosely ordering them by the extent to which they can meet the different bounds.

Table 4.2: Comparison of assumed sending behavior

Protocols	Dummy traffic per round & user	Communications per round	Counting-Bound
Herd	1 (some)	n (or more)	X
DC-Net, Dissent	1	1	✓
Dicemix	n	n	✓
Vuvuzela	1	n	X
Riffle, Riposte	0 (1)	n	X

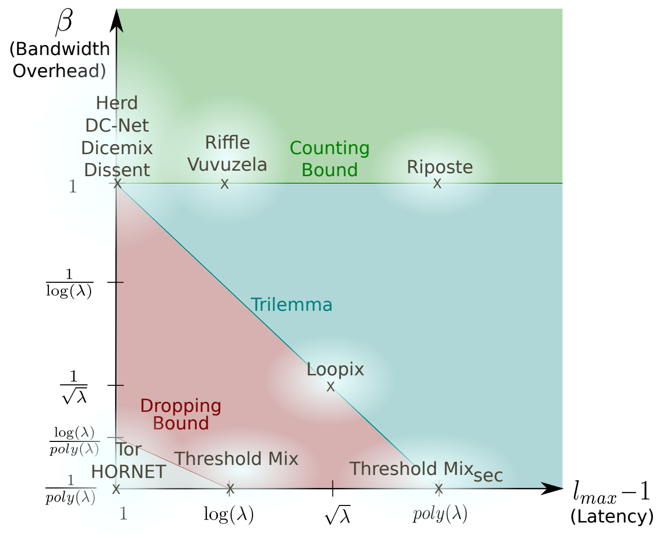


Figure 4.3: Comparison of bounds under the special set of assumptions of the Trilemma [54] (see Table 4.2 for the Counting-Bound in the general case). To depict all bounds in one figure, we need to assume the most strict assumptions: the assumptions of the Trilemma. As [54] we assume that $\beta \approx p$ to summarize both user settings. The Counting-Bound requires the highest overhead but is independent of the latency. The Trilemma shows a trade-off between latency and bandwidth, it is higher than the Dropping-Bound.

4.3.1 Discussion of Networks

Figure 4.3 illustrates the trade-off vs. the cost of different existing ACNs. We facilitate this comparison by restricting ourselves to a specific scenario: A single real message is assumed to be sent during each round. The abscissa denotes the latency of messages, and the ordinate the bandwidth overhead, as part of the probability that a node sends a message during a round.

We further discuss the more general cases, which are especially interesting to assess the systems according to the Counting-Bound – and we give an overview on the assumptions of the different protocols necessary to assess this in Table 4.2.

Tor [58], HORNET [43]. This first class of low overhead onion routing systems sends messages over a path of relays and does not employ additional dummy traffic. The number of hops is fixed, and thus they expose constant latency.

These systems fall short of any bound. All explained attacks indeed are successful (or expected to be): It is simple to link sent onions because of their timing (Trilemma), to count the number of messages a sender sent as sending can be observed (Counting-Bound). Knowing that a certain receiver expects another packet (e.g., because the use case postulates a message stream), dropping it right at the sender can be recognized at the receiver (Dropping-Bound). This in itself is not new, and corresponding attacks have been suggested [119] or are at least conceivable for Tor, which is build for a weaker adversary model.

Threshold-Mix [140]. This class of mixes collects t messages before relaying them further. It does not employ dummy traffic. Thus, each sending event transmits a real message, and $S\overline{O}$ cannot be achieved according to the Counting-Bound. Interestingly, the approach can however fulfill the two other bounds: If each user sends one message and each mix waits for all of them, and if further all mixes are used (as assumed if a high latency is allowed), the attacks fail. Dropping a message yields no message to be delivered, and hence the privacy is kept (although availability is jeopardized). As long as we assume that one of the mixes is honest, linking the incoming and outgoing packets fails at this point and also timing does not provide any help as the first mix already waits for all messages. We do however agree with [54] that the Trilemma cannot be met for convenient thresholds and numbers of mixes (lower than in the order of the number of users).

Herd [104], DC-Net [40], Dicemix [137], Dissent [156]. This class of systems employs dummy traffic but has low latency. Herd uses multiple relays just like Tor and HORNET, but adds dummy traffic. DC-Net, Dicemix and Dissent in contrast follow the idea of secret sharing. They generate the original message as a combination of both; a real message from one and dummy messages from all other users.

Only the Counting-Bound is applicable to these secret sharing based systems, as both the Dropping-Bound and the Trilemma are based on the mixing model.

The systems indeed meet the overhead requirement of the Counting-Bound. Without a collision avoiding scheme DC-Nets still cannot achieve the notion $S\overline{O}$ (cf. [72]). Dicemix and Dissent specify scheduling for transmission slots by combining one message of each user in every round. Mounting the attack from the Counting-Bound, the receiver hence does learn that all messages of a single round are from different senders, and only messages distributed over multiple rounds can be from the same sender. She succeeds and the notion $S\overline{O}$ cannot be achieved in consequence.

The situation for Herd is a bit more complex, than the representation in Figure 4.3 suggests. The graph assumes only a single communication per round, and for this special case Herd meets all bounds as it employs enough overhead. However, Herd aims at a VoIP scenario, which indicates that the more general case of users participating with several communications in the same round seems more applicable. The Counting-Bound is no longer met in this case: The users in Herd generate a predefined amount of traffic, which is supposed to at least resemble the traffic caused by a small number of VoIP connections (e.g. one). This does not outweigh the total number of real messages sent during a round, and the Counting-Bound is violated. Herd in consequence leaks some information about the sender behavior and $S\overline{O}$ cannot be achieved.

Loopix [128]. Loopix is another mix network that adds more mixes and dummy traffic. It allows to adjust both the number of used mixes and the dummy traffic via parameters. Sender traffic is generated according to an exponential distribution. Like [54] we assume $\sqrt{\lambda}$ mixes per path and dummy traffic with probability $\frac{1}{\lambda}$, although we stress that other parameter choices are not excluded by the paper.

Loopix in this setting satisfies nearly all bounds. Only from the Counting-Bound, we can conclude that it cannot achieve $S\overline{O}$. We expect that also practically the two scenarios of either one user sending many messages, or the same number of messages being sent by multiple users can be distinguished: The messages in the first case arrive much slower at the receiver. However, aiming at $S\overline{O}$ may be too strong for many use cases and a weaker privacy notion targeted. Loopix does employ protection measures against the other attacks. Confirming their effectiveness is however beyond the scope of this work.

Riposte [49]. Riposte uses a reversed PIR to implement an anonymous broadcast. Each client sends a message to the PIR servers in the epoch during which she participates. Riposte does not apply the concept of dummy messages[12]. The set of senders is published at the end of each

[12]Only for receiving an empty message is used, as messages in the postboxes of the clients are swapped.

round.

Riposte does not lend itself to analysis with the model of the Trilemma, as the latter assumes only a single sender to send a real message per round, but Riposte requires several parallel communications to achieve any anonymity. In Figure 4.3 we still follow [54] and choose the probability for every sender to send in each epoch to be one. The categorization with the assumptions of the Trilemma is however misleading for the general case in Riposte as not only one, but multiple messages are sent per epoch. Similar to Herd, the bandwidth overhead is again too small to withstand the requirements of the Counting-Bound. We can confirm this theoretically with an attack: By observing the number of write requests to the servers (i.e. send events), an adversary can directly count the number of sent messages, as no dummy traffic is applied. Riposte clusters sending events, so they are not spread over several rounds, and they are only hidden among each other. Further, although the latency is sufficient to fulfill the Dropping-Bound, the dropping attack still works: Dropping all parts of the write request of one user will not lead Riposte to stop, but instead to publish all except this user's message.

Riffle [103], Vuvuzela [154]. Vuvuzela and Riffle are mix networks that require all messages to go trough all mixes. Alike [54], we assume a logarithmic number of mixes. Further, Vuvuzela ensures a constant traffic rate by employing dummy traffic. Riffle assumes all clients to always have a message to send ("each client onion encrypts a message"). So, in both protocols each client sends in every round. Riffle additionally employs PIR to deliver the messages after they went through a verifiable shuffling mix net.

They intuitively seem to satisfy all bounds and could possibly achieve all notions. Similar to Herd however, multiple users can (Vuvuzela) or have to (Riffle) send every round and we can infer that all messages of one round have been sent by different senders. Thus the Counting-Bound is only fulfilled for the special case that only one user sends per round. This case might happen, but is not enforced in Vuvuzela, and contradicts the assumption of Riffle that each client sends a message.

4.3.2 Final Remarks on Bound Implications for Protocol Proposals

The bounds show limitations of existing ACNs, as they cannot achieve certain privacy notions. We managed to underline this situations with attacks on the systems. We also conclude that nearly no system achieves $S\overline{O}$ nor reaches the Counting-Bound under the given assumptions. It turns out that the assumption of the number of real messages sent per round is important, not only to assess specific bounds and check for their applicability in the first place, but also to put the bandwidth overhead into perspective.

We note that there are cases where we suspect that the protocols do not achieve certain privacy goals even though they reach the corresponding bounds.

4.4. A Practical Viewpoint: Explaining Limitations

Arriving at this bleak outlook, we want to put the bounds into perspective.

4.4.1 Strong Privacy Goal Formalizations

The Notion $\overline{SO}$ of the Counting-Bound

$\overline{SO}$ is a strong notion[13], which even hides the number of active senders. While there are use cases for this notion (see Section A.6), for many proposed protocols it might be too strong. Some protocols (cf. Section 4.3) aim however to protect against a similar, but weaker notion: They ensure that any user sends a fixed, small number of communications (real or dummy) every round. Thereby, they allow the adversary to learn that no user has sent more than this number of real messages, which implicitly leaks a lower bound on how many senders have been active during a given round.

Relaxed Notion $\overline{SO}_\mu$. The notion ensures that each sender at most wants to send some small number μ of messages per round. Thus it hides whether the frequency is 0,1,2, ... or μ, but if a sender desires to send more than μ messages per round this can be observable, as some of the expected messages only arrive in a later round. With smaller μ, frequencies are less protected and more about the intended sending frequency can be deduced from the number of rounds until the delivery of the message.

Formally, with the definition of Q_b from Chapter 3 we can define the requirement of the relaxed notion $\overline{SO}_\mu$ as follows: The batches are valid iff for all j: $r_{1_j} = (\mathbf{u_{1_j}}, u'_{0_j}, m_{0_j}, aux_{0_j})$ and for $b \in \{0,1\}$ for all $(u,n) \in Q_b$: $n \leq \mu$.

Game-Based Notions for Bounds

Everything that *could* leak in the protocol by definition of the game-based notion is assumed to *be* leaked during the analysis. This is useful for worst case analyses. For bounds, however, the adversary knows, per game definition, everything that happens as long as it is not explicitly defined to be protected. She does not even have to be able to observe any of this in reality.

Consider the Optimality-Bound: The adversary knows how many real messages are received, without controlling the receiver, just by the definition of the notion. Further, the attack in the Dropping-Bound requires her to realize that a packet is missing. In the game-based notion, this is trivial: the adversary chooses how many messages each receiver expects, by the definition of the notion. In reality, this limits the applicability to use cases with predictable receiving behavior (like streaming or triggering the reaction with rumor spreading).

Future work on bounds should therefore argue the practicability of the underlying attack and assumptions. For more realistic analyses communications unknown to the adversary and beyond her control could be included[14].

4.4.2 Maximal Anonymity Sets

All bounds require the anonymity set to include all users and that even the considered attack cannot exclude a single user from it. For many real use cases, however, significantly smaller anonymity sets after an attack may be sufficient. For example, building the anonymity set only from the users concurrently online (or sending) might be acceptable for the use case as long as at any point *enough* users are online (or sending). Determining such suitable smaller anonymity sets will be a challenge for future work.

[13]Note, that also much stronger notions, which require for instance membership concealment, hiding the fact if a user participates in the system at all, are discussed in literature.

[14]This extension is easily achieved by adding adversary classes (see Section A.1.4)

4.4.3 Bandwidth Cost Models

Different concepts are summarized under the term “bandwidth overhead”. For the Trilemma bandwidth overhead naturally occurs from dummy messages, while for the Dropping-Bound redundant copies of the real messages are needed. Further, also for dummy messages end-to-end dummy traffic, starting and ending at users, and link dummy traffic, which is just applied to obscure the traffic on one hop, exist.

Interestingly, the overhead in the Counting-Bound and Trilemma measures only sender-generated dummy messages. In practice, however, end-to-end dummy traffic puts more load on the network than link dummy traffic at the sender’s first link. In the Trilemma, for example, longer lasting dummy traffic is only necessary if corrupted relays are introduced into the model. Contrary to the cost definition of the Counting-Bound and Trilemma, the Dropping-Bound’s can reflect a difference between end-to-end dummy traffic and dummy traffic on the first link. We thus prefer this cost metric for future work.

4.4.4 Assumptions

Relaxed assumptions are desirable for future work on bounds to improve their applicability. In terms of sending behavior, having more than a single user send a real message per round, contrary to the Trilemma’s assumption, suits reality better and naturally benefits the privacy. Further, latency requirements, like in the Trilemma, may further be more relaxed in many practical use cases.

5. Onion Routing and Mix Message Formats: Breaking and Fixing the Unidirectional Case

During our investigation of state-of-the-art protocols to identify basic building blocks, we discovered an attack on a series of onion routing and mix networks, which we will discuss in detail as part of this chapter. Motivated by this attack, we use the next two chapters to investigate Onion Routing (OR) and mix network protocols. Our aim in these chapters are secure packet formats. Consequently, attacks based on timings and traffic patterns are out of the scope, as the packet format itself cannot hide such information. However, real world protocols of course should prevent linking based on timings with additional measures, as well as employ a secure packet format.

To be compliant with the terms of the investigated related work [31], we understand OR in this context as a free-route mixnet [42] without requiring that messages are delayed. This conforms with the understanding of typical onion routing networks [58, 74] except that circuits are excluded. Note that even though our proofs and packet formats are using the terms of OR, they are of course also suitable for mix networks.

This first chapter on OR will focus on the unidirectional case, while the next explains and solves the challenges introduced by allowing replies from the receiver back to the anonymous sender. Results of this chapter have been published at IEEE S&P 2020 [95] (and in the corresponding extended version [94]).

5.1. Background

This section introduces Onion Routing (OR), mix networks and their formal state of the art as well as related protocols in more detail. 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 to hide from a regime that deploys mass surveillance.

5.1.1 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 PPT algorithms and does not control *all* nodes. These assumptions are common for onion routing.

5.1.2 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; 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.* Onion Routing can be used in both network modes (see Section 2.1.1): the *integrated system* model, where the receiver as the last router of the path gets an onion, and the *service* model, where last router, the *exit node*, forwards only the final message to the receiver.

To enable the layered encryption, we assume the existence of public keys PK for all relays.
Assumption 1. *The sender knows the (authentic) public keys PK_i of all relays P_i it uses (e.g., by means of a public key infrastructure (PKI)).*

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 2. *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 3. *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 4. *The replay protection, if one is used, drops bit-identical onions.*

5.1.3 Existing Schemes and Systems

Danezis and Goldberg [52] 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

Table 5.1: Notation Summary

Notation	Meaning
$\|\|$	concatenation of strings
λ	the security parameter
$\mathcal{P}$	an onion path
m	a message
P_i	for the i-th router name on the path, P_0 is the sender and P_{n+1} the receiver
PK_i	public key of P_i
SK_i	private key of P_i
O_i	$= (\eta_i, \delta_i)$ is the i-th forward onion layer to be processed by P_i
η_i	the header of O_i
δ_i	the payload of O_i
FormOnion	the function to build a new onion as a sender.
ProcOnion	the function to process an onion at a relay.

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 [31] for Sphinx.

Two protocols [53, 142] preceded Sphinx and a series of networks [43, 44, 128] builds upon it. We will introduce these works later, as needed.

5.1.4 Formally treating OR

Early on, Mauw et al. [111] modeled and analyzed OR. However, this work does not contain any proposal to prove future systems secure. Backes et al.'s ideal functionality [9] models Tor and hence includes sessions and reply channels. However, it is very specific to Tor and thus rather complex and can hardly be reused for general onion routing and mix networks. The Black-Box Model of Feigenbaum et al. [66] uses a very high-level approach, but does not support replies. Further approaches [36, 37, 87] propose some security properties, but do not give any ideal functionality or similar concept that would allow to understand their concrete implications for the users' privacy.

As the most prominent formalization without replies, Camenisch and Lysyanskaya [31] defined an ideal functionality in the UC-Framework (see Section 2.3.3) and showed properties an onion routing protocol needs to fulfill to prove its security. Proving these properties has become the preferred strategy to prove mix and onion routing networks secure. Although the strategy is designed for the integrated system model, it applies to the service model as well (except for renaming *recipient* to *exit node*). In the service model 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.

The strategy has been used to prove the correction [142] of Minx [53], as well as for the security proof of Sphinx [52]. The most eminent, recently proposed practical onion routing and mix networks [43, 44, 128] build on Sphinx as packet format to subsequently tackle traffic analysis and timing attacks, while proving the security of their adaptions to Sphinx still based on the properties of Camenisch and Lysyanskaya.

Formal OR Scheme. We use the notation from Table 5.1 for the description of the formal OR scheme. To model OR, [31] defines an *Onion Routing Scheme* as in the following definition:

Definition 14 ([31]). *An* OR Scheme *is a tuple of PPT algorithms* $(G, \text{FormOnion}, \text{ProcOnion})$*:*

Key generation: $(PK, SK) \leftarrow G(1^\lambda, p, P)$ *with public key* PK*, secret key* SK*, security parameter* λ*, public parameter* p *and router name* P

Sending: $(O_1, \ldots, O_{n+1}) \leftarrow \text{FormOnion}(m, (P_1, \ldots, P_{n+1}), (PK_1, \ldots, PK_{n+1}))$ *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: $(O', P') \leftarrow \text{ProcOnion}(SK, O, P)$ *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.*

Ideal Functionality $\mathcal{F}$. [31] proposes an ideal functionality for OR that expresses what a private and secure OR scheme is allowed to reveal to an adversary.

To understand the ideal functionality, 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 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 description of [31] with highlighted small changes, as well as a pseudo code representation of the ideal functionality is in Appendix C.1.1.

As there is confusion about the protection provided by the ideal functionality (see Appendix C.1.3), we analyze it in Appendix C.1.3 to find that it is restricted to the cryptographic properties of onion routing and does not protect against timing and traffic-analysis attacks, like dropping or delaying onions. Hence, for our analysis we exclude attacks that result in dropping or delaying onions. 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).

Given this adversary model, we are able to prove the privacy goals expected for OR. Especially, Sender-Message Unlinkability ($SM\overline{L}$) is achieved even if the receiver is corrupted. From our hierarchy (Figure 3.3 of Section 3.5), we know that this notion includes some protection for the sender-receiver relationship ($(SR)\overline{L}$) as well. Even if in practice stronger adversary models are assumed, proving the realization of the ideal functionality is a useful way to reduce the problem of proving privacy to the attacks excluded by this adversary model.

Properties. [31] defines three *security properties* for OR schemes and proves that those imply realizing their ideal OR functionality. Later works [43, 44, 52] 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 adversarially 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.

5.2. 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.

5.2.1 Attack on Sphinx

In Sphinx as presented in [52] 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 that were 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[1] 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.

5.2.2 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

[1]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`

to the proposed combination of properties, at all. Onion-Security is required, but fails to protect against some weaknesses.

Onion-Correctness

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

Definition 15 (Onion-Correctness [31]). *Let* $(G, \text{FormOnion}, \text{ProcOnion})$ *be an OR scheme with 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*

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

the following is true:

1. *correct path:* $\mathcal{P}(O_1, P_1) = (P_1, \dots, P_{n+1})$,
2. *correct layering:* $\mathcal{L}(O_1, P_1) = (O_1, \dots, O_{n+1})$,
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.

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. The full argument that can be found in [94] is rather extensive and hence omitted from this thesis. 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.

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.

Game Structure We illustrate the Onion-Security game in Fig. 5.1 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,

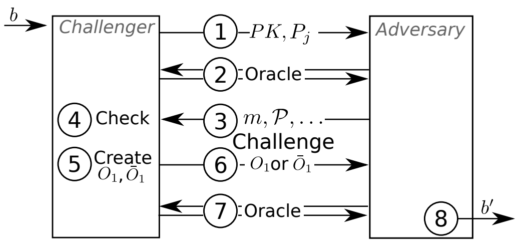


Figure 5.1: Onion-Security game illustrated: Circled numbers represent the steps, and the boxes the parties of the game.

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 $O_1, \bar{O}_1$, which we detail below, 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 game 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. 5.2. 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.

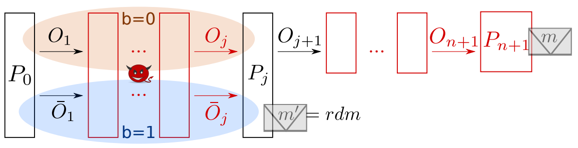


Figure 5.2: Cases of Onion-Security illustrated: Red boxes represented corrupted relays, black boxes honest. The upper row of arrows represents the path of the onion O with inputs chosen by the adversary (message m received by P_{n+1}); the lower an onion $\bar{O}$ containing a randomly chosen message 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.

Definition Formally, the Onion-Security game is defined as follows:
Definition 16 (Onion-Security [31]). *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* 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:*
   $(O_1, \dots, O_j, \dots, O_{n+1}) \leftarrow \text{FormOnion}(m, (P_1, \dots, P_{n+1}), (PK_1, \dots, PK_{n+1}))$
   $(\bar{O}_1, \dots, \bar{O}_j) \leftarrow \text{FormOnion}(m', (P_1, \dots, P_j), (PK_1, \dots, PK_j))$

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* $\text{ProcOnion}(SK, O_i, P_j)$.

8. *The adversary then produces a guess b'.*

Onion-Security is achieved if any 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.

5.2.3 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.

Insecurity: Signaling the Path. We first generalize the attack on Sphinx from Section 5.2.1, which only leaked the path length. As generalization we give a protocol idea that complies to the properties, but includes the complete path (including the sender) in the message. Thus, an adversarial receiver learns the complete path that 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. As the given properties, however, never return the information that an adversarial receiver learns to the game adversary, they cannot protect against this and similar weaknesses. (See Appendix C.3.1 for such an insecure protocol.)

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 5.2.3 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.

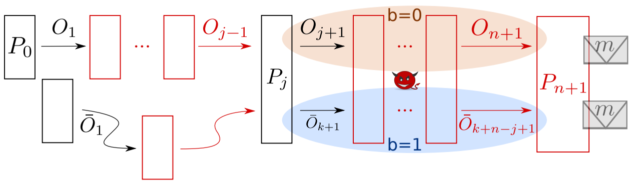


Figure 5.3: Cases of TI illustrated: Red boxes are adversarial routers; black boxes honest and curvy arrows symbolize a random path possibly through many other adversarial routers. In case $b = 0$ the adversary chosen onion is observed at the end of the path (orange ellipse). For $b = 1$ onion layers 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.

Intuitively, the steps are the same as in Onion-Security described in Section 5.2.2, 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 ($b = 1$). We illustrate the new outputs to the adversary in Fig. 5.3 and formally define the new property in our Definition 17.

Thus, our first new property TI is defined as:
Definition 17 (Tail-Indistinguishability TI)**.** *Consider an adversary interacting with an OR challenger as follows.*

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 $\frac{1}{2}$.

5.2.4 Linking Protection

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

Insecurity: Including Unique Identifiers. We demonstrate a second weakness by introducing a fixed identifier for each onion that is included in any layer of this onion. Thereby the onion layers of the same onion, as well as the corresponding communication partners (and if the receiver is corrupt the message), can be easily linked. This and similar weaknesses are not detected by the given properties as they only output information before or after the honest relay, but never both. (See Appendix C.3.2 for such an insecure protocol.)

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. 5.4.

Note that this new property indeed prevents the insecurity given in Section 5.2.4: 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 5.2.2 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. 18.

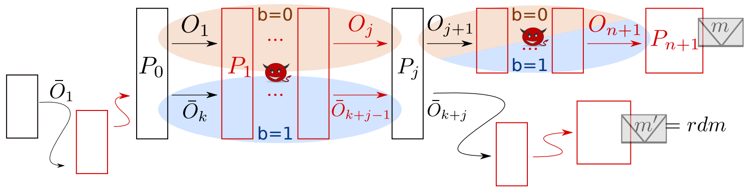


Figure 5.4: Cases of LU illustrated: Red boxes are corrupted routers, black honest routers and curved arrows represent randomly chosen paths. In case $b = 0$ the adversary chosen onion, sent from P_0, is observed on the complete path $\mathcal{P}$ starting from when it arrives at the first relay P_1. 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 18 (Layer-Unlinkability LU). *1. – 4) as in Def. 17*

5. *The challenger creates the onion with the adversary's input choice:*

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

and a random onion with a randomly chosen path $\bar{\mathcal{P}} = (\bar{P}_1, \dots, \bar{P}_k = P_1, \dots, \bar{P}_{k+j} = P_j, \bar{P}_{k+j+1}, \dots, \bar{P}_{\bar{n}+1})$*, 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* $m' \in \mathcal{M}$*:*

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

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* $\frac{1}{2}$*.*

5.2.5 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. 5.5.

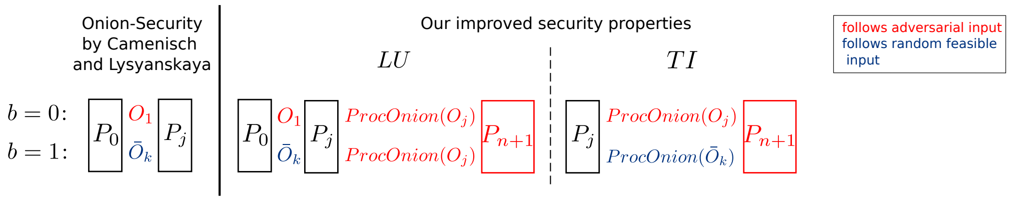

Onion-Security by Camenisch and Lysyanskaya
Our improved security properties
LU
TI
follows adversarial input
follows random feasible input
$b = 0$:
$b = 1$:
P_0
O_1
$\bar{O}_k$
P_j
ProcOnion(O_j)
P_{n+1}
ProcOnion($\bar{O}_k$)

Figure 5.5: 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.

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 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 layers of a random replacement onion without her noticing it. The replacement onion only corresponds with the original onion in characteristics that 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. The key distribution itself is however outside the scope of this work.

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 19. *A secure OR scheme is a triple of polynomial-time algorithms (G, $FormOnion$, $ProcOnion$) as described in Section 5.1.4 that achieves Onion-Correctness (Definition 15), Tail-Indistinguishability (Definition 17), as well as Layer-Unlinkability (Definition 18).*

Following Camenisch and Lysyanskaya, we build a protocol from any secure OR scheme by using another ideal functionality for the distribution of the public keys. Let therefore $\mathcal{F}_{RKR}$ be the ideal functionality that allows to register the public keys.

Definition 20. *OR protocol Π is a secure OR protocol (in the $\mathcal{F}_{RKR}$-hybrid model), iff 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 $(SK_i, PK_i) \leftarrow G(1^\lambda)$ and publishes PK_i by using $\mathcal{F}_{RKR}$.*

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

Processing an Onion: *P_i received O_i and runs $(O_j, P_j) \leftarrow \text{ProcOnion}(SK_i, O_i, P_i)$. 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 C.2.2.

Theorem 6. *A secure onion routing protocol following Definition 46 UC-realizes $\mathcal{F}$ in the $(\mathcal{F}_{RKR})$-hybrid model.*

5.3. Second Pitfall: Insufficient Oracle Treatment

During our investigation of state-of-the-art network proposals, we discovered another attack on the onion routing network HORNET [43], that 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 describing our malleability attack on HORNET (Section 5.3.1), then we explain how the oracles have been modified and how the correct use of oracles detects the attack (Section 5.3.2).

5.3.1 Malleability Attack

HORNET [43] 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.

The authors claim that HORNET protects the anonymity of a sender 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 the routing of data in the subsequent transmission phase. Multiple cryptographic primitives are used in the construction and processing of packets in both phases. More precisely, HORNET uses a pseudo-random permutation (PRP) in CBC mode[2] to form layered encryption of its payload, but does not implement integrity checks at the processing relays for it.

An attacker that controls the first relay[3] and the receiver can link sender and receiver (thus break $(SR)\overline{L}$) and this adversary can also link large parts of the message to its sender (break $(SM)\overline{L}$) with the following malleability 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.

[2]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.

[3]Technically, controlling the link from the sender to the first relay is enough. However, whether the adversary controls links is not explicitly stated in [43].

Varying the Attack for Related Systems

The attack can be used on related works and the setup phase of HORNET as well. See Table 5.2 for a summary. The privacy of all these protocols and protocol phases is proven following the same proof technique from Camenisch and Lysyanskaya [31].

Sphinx. Sphinx specifies that headers and payload are encrypted independently of each other. The payload is encrypted using a bidirectional error propagating block cipher and protected with an integrity check for the receiver, but not for processing relays. Further, Sphinx model considers the receiver to not be part of the anonymization protocol. The receiver's address is included in the payload.

Because of the block cipher choice, our attack destroys the payload completely, and the sender cannot be linked to the original message content and, if used in the intended model, the sender can only be linked to the exit node.

However, if Sphinx is used with the receiver as the last relay (no address needs to be coded in the payload), it remains possible for a corrupt receiver to determine, who attempted to communicate with her by using our attack. The receiver does not even need to be able to distinguish a real message from random bits, but just has to notice that Sphinx's integrity check at the receiver failed.

Loopix. *Loopix* [128] builds on Sphinx and contributes a new dummy traffic and sending strategy. The Sphinx header, as used in Loopix, includes the path until the receiver[4]. In the payload only receiver.host, receiver.port, receiver.name get addressed again. This means the receiver of the modified packet will receive a message with random-like content. Random payload is used for dummy messages in Loopix. Note, however, that the receiver gets only two types of dummy messages: i) dummy from the provider that is tagged with 'Dummy'[5] and ii) loop dummy messages that the receiver initiated. Hence, the receiver can recognize whether the received random message is unusual and the malleability attack works. The attack further lies within Loopix adversary model and allows to link the sender and receiver, but destroys the message.

HORNET's Setup Phase. HORNET's setup phase uses Sphinx in the setting that the last relay is the receiver, and hence the linking of sender and receiver is possible.

TARANET. *TARANET* [44] extends HORNET to protect against partially stronger adversaries. TARANET bases its setup on Sphinx as well. Additionally, it proposes packet-splitting as a traffic-shaping technique to withstand traffic-analysis. Therefore, however, shared trust between sender and receiver is presumed. TARANET's adversary model hence excludes our attack. For the sake of completeness, we argue the effect of our attack when the trust assumption is violated: TARANET uses HORNET's Sphinx-based setup phase and thus is prone to the attack in this phase. Its data transmission phase however protects the integrity of the complete onion at each hop. Thus, in this phase the attack fails.

Improved Minx. Predecessors to Sphinx were *Minx* [53] and its fixed version [142]. Like Sphinx, neither of the two protects the integrity of the payload at the relays. Minx nodes hence process all onions they receive. Thus, our attack to link sender and receiver works under the assumption that

[4] see `https://github.com/UCL-InfoSec/loopix/blob/440db95fdd2bcb28d76aca1b86f072715ef7005a/loopix/loopix_client.py#L149`,
`https://github.com/UCL-InfoSec/loopix/blob/440db95fdd2bcb28d76aca1b86f072715ef7005a/loopix/core.py#L47`,
`https://github.com/UCL-InfoSec/sphinx/blob/master/sphinxmix/SphinxClient.py`)

[5]`https://github.com/UCL-InfoSec/loopix/blob/440db95fdd2bcb28d76aca1b86f072715ef7005a/loopix/loopix_provider.py#L81`

Table 5.2: 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 $\neq$ exit node)			✓
Sphinx (receiver = exit node)[6]		✓	✓
Loopix		✓	✓
HORNET (Setup)		✓	✓
TARANET (Setup)		(✓)	(✓)
HORNET (Data)	✓	✓	✓
TARANET (Data)			

the receiver can distinguish valid messages from random bits. Similar to Sphinx, both Minx and the improved version employ bidirectional error propagating block ciphers, so recovering (parts of) the message after modification is impossible. In contrast to Sphinx, the improved Minx's network model already allows for our attack.

Ramifications

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. It also lies within the adversary model of the improved Minx as it is an active tagging attack on the first link that allows to discover the destination. Further, as discussed earlier, it lies in the adversary model of Loopix as well. For Sphinx and Taranet the deviation from their network resp. adversary model seems to explain the existence of this attack.

Loopix relies on the security of Sphinx and does not prove any additional onion routing properties. The existence of this attack in HORNET and the improved Minx however contradicts their security proofs. The reason for this are not the flaws in the properties of [31], but a common mistake in proving the properties.

5.3.2 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 [31] 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 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 5.3.1) 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 5.5.3).

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

5.4. Proving the Adapted Sphinx secure

Sphinx specifies to use a header and a payload. The original Sphinx [52] 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 5.3.1 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 5.2.1 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 [14]. 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 5.2.1 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 [52]. 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.3.4, and it leads to the following theorem:

Theorem 7. *Beato's Sphinx variation, adapted with the fix to the attack from Section 5.2.1, 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 our Sender-Message Unlinkability ($SM\overline{L}$) that includes protection of the sender-receiver relationship ($(SR)\overline{L}$) 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.

5.5. Discussion

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

5.5.1 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 FormOnion, as the adversary could use 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.

5.5.2 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 (see Appendix C.3.5 for details).

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).

5.5.3 Limitations

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

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.

Reply Channels and Sessions. All systems that proved privacy with the properties consider a reply channel to respond to the 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, the properties, as well as the ideal functionality need to be adapted for this case in future work.

6. Provably Secure Onion Routing with Replies

In this chapter, we tackle the challenge of repliable onions for bidirectional communication, as compared to the unidirectional case in the last chapter. We provide the adaption to the security definitions, as well as introduce two secure packet formats that achieve our security definitions. As in Chapter 5, we limit the scope to secure packet formats and thus do not discuss timing and traffic analysis attacks. Results of this chapter have been published at Asiacrypt 2021 [97] (and in the corresponding extended version [98]).

6.1. Overview

Most natural use cases for Internet communication are bidirectional, i.e., require a receiver to respond to the sender. However, in an OR-transmitted message the receiver has no obvious way to send a *reply* back to an anonymous sender. Note that adding a sender address in plain to the payload message would of course defeat the purpose of OR (as discussed in Chapter 5). Even encrypting the sender address, say, with the public key of the receiver (so that the receiver can use another OR communication to reply), is not appropriate as we may not always trust the receiver to protect the sender's privacy (e.g., like a newspaper agency being forced to reveal whistleblowers).

Perhaps surprisingly, this problem of "OR with replies" has not been formally addressed in the OR literature with sufficient generality (with one recent insufficient exception that we will refer to below). Hence, our goal in this work is to provide definitions and instantiations for OR protocols "with an anonymous back envelope". That is, we attempt to formalize and construct OR protocols which allow the receiver to reply to the sender *without* revealing the identity of the sender to anyone.

Recall that in Chapter 5 we recognized and corrected flaws in the definitions and application of earlier work [31] that allowed for sincere practical consequences in the form of a *malleability attack* (see Section 5.3.1). While the corrected properties are sufficient for their setting, Chapter 5 only partially fixes the situation as the models do not include support for reply messages (to the anonymous sender). Sphinx and the improved version of Minx make adaptions to the properties of Camenisch and Lysyanskaya to account for replies to some extent, but thereby they build on the flawed properties and do not treat replies correctly, thus limiting the achieved privacy. Even worse, nearly all of the eminent recent network proposals claim to support anonymity for replies, while relying on the flawed properties. In a recent work [5], Ando and Lysyanskaya define an ideal functionality for repliable onion routing. They also propose corresponding properties and a protocol that satisfies their ideal functionality. In this, they however accept vulnerability against the malleability attack in their definition of security, as well as in their protocol. Therefore, the question of a *secure*, repliable OR scheme is still unanswered.

A technical challenge with practical relevance. The goal we are aiming at is not only useful, but also technically difficult to achieve. First of all, practically prominent protocols and packet formats [43, 52, 142] require that *any reply is indistinguishable from any forward request*, except at the sender and receiver. In particular, all parts of the onions in both directions should look alike, and must be treated according to the same processing rules. This is necessary to provide senders that expect replies with a sufficiently large anonymity set even if there is only a small amount of reply packets, because they are hidden under all forward traffic.

To prevent the malleability attack, tampering by a potentially corrupt relay must become detectable. Theoretically, conventional payload authentication for all forward layers, e.g., with MACs precalculated by the sender, is sufficient. However, both Ando and Lysyanskaya [5] and practical proposals [43] require indistinguishability of forward and reply onions. Extending authentication also to the reply payload is challenging: The original sender cannot precalculate those authentication tags as the reply payload is unknown and we cannot necessarily assume that the receiver is honest. Letting the reply sender (= original receiver) precalculate the authentication tags enables an attack similar to the malleability attack: the malicious reply sender (= receiver) together with the last relay can recognize the reply onion (without modifying its payload on the way) simply based on the known authentication tags; thus letting the attacker learn the same metadata as in the malleability attack. Hence, payload protection in the reply setting is the real challenge towards a practical solution.

Overview of the next Sections. We present a framework for repliable OR, along with two different instantiations (with different properties). Our framework protects against malleability attacks on the payload, while even guaranteeing that replies are indistinguishable from original requests. In our approach, hence, both requests and replies are authenticated *implicitly* (i.e., without MACs) at each step along the way.

From a definitional point of view, we express these requirements by an ideal functionality (in the UC framework) which does not reveal the onion's path, message or direction to the adversary (unless all involved routers are corrupt; further a corrupt receiver of course learns the message and direction). This translates to strong game-based properties, which are proven to imply the security in the sense of that ideal functionality.

We also present two protocols that realize this ideal functionality. Both of our OR protocols are in fact similar to existing protocols, and are partially inspired by the popular Sphinx approach [52] and the Shallot scheme of Ando and Lysyanskaya [5]. The main conceptual difference to previous work is that the authentication of the (encrypted) payload happens *implicitly* in our case.

Our first protocol uses updatable encryption (UE), a variant of symmetric encryption that provides both rerandomizable ciphertexts (and in fact RCCA security [33] and plaintext integrity) and rerandomizable keys, as a central primitive. Intuitively, using UE for encrypting the payload message (in both communication directions) enables a form of "implicit authentication" of ciphertexts, and hence thwarts malleability attacks without explicit MACs on the payload.

Our second protocol is based on succinct non-interactive arguments (SNARGs [113, 24]), a variant of zero-knowledge arguments with compact proofs. Intuitively, SNARGs enable every relay *and* the receiver to prove that they have processed (or replied to) their input onion according to the protocol. This way, no explicit authentication of the payload data is necessary, since the SNARGs guarantee that no "content-changing" modification of the payload took place.

Neither of our protocols indeed are competitive in efficiency with existing OR protocols (see Appendix C.4.4 for details on the performance of the protocols). Further, our protocols require a trusted setup. This is due to the introduction of new concepts and techniques for qualitatively stronger security properties. Our work however represents an important conceptual first step towards an efficient *and* secure solution.

6.2. Notation

We use and extend the notation of Table 5.1 from Section 5.1.4. Additionally, we use ReplyOnion as the function to reply to a received onion as receiver. Further, we use the superscript $x^{\leftarrow}$ to denote the corresponding entity on the backwards path. For example, while P_1 is the first router on the forwards path, $P_1^{\leftarrow}$ is the first router on the backwards path. Typically the forward sender P_0 is also the backward receiver $P^{\leftarrow}_{n^{\leftarrow}+1}$ and the forward receiver P_{n+1} is the backwards sender $P_0^{\leftarrow}$.

6.3. Model and Ideal Functionality

We first define our assumptions and model for repliable OR and then describe our desired security as the ideal functionality. Our model extends the OR scheme definition of [31] as used in Chapter 5 by adding an algorithm to create replies. Our ideal functionality extends the one of [31] as used in Chapter 5 and has similarities to [5], but is strictly stronger as it requires protection against malleability attacks on the payload.

6.3.1 Assumptions

We make the following assumptions that result from commonly used techniques to ensure unlinkability of onion layers on criteria other than the concrete representation of the onion. We re-state Assumption 1 and 2 of Section 5.1.2 as we need them again. Further, we introduce adapted duplicate protection assumptions to support the reply case (as compared to replay protection before).
Assumption 1. *The sender knows the (authentic) public keys PK_i of all relays P_i it uses (e.g., by means of a PKI).*
Assumption 2. *The protocol's maximum path length is N.*

To ensure that packets cannot be linked based on the included routing path, the sender includes the routing information encrypted for each forwarder, such that any forwarder only learns the next hop of the routing path. We assume that the routing information is included in a header, while the message is included in the payload of the onion.
Assumption 5. *Each onion O consists of a header η and a payload δ.*

To ensure that packets cannot be linked based on duplicate attacks, i.e. the onion of the victim is duplicated at the first corrupted relay and observed twice at the corresponding receiver, duplicates have to be detected and dropped. We support duplicate detection with deterministically evolving headers, which allows to also protect from duplicated replies. Thus, even though the (forward) receiver is allowed to decide on her answer arbitrarily, we can still detect if she tries to send multiple different answers to the same request. As some related work wrongly adapted proof strategies for OR schemes where the duplicate detection is solely based on *the header of the onion*, we deliberately build this model for OR-schemes allowing for such protocols.
Assumption 6. *Duplicates, i.e. onions O_i, O'_i with the same header $\eta_i = \eta'_i$, lead to a fail for every but the first such onion that is given to* $\text{ProcOnion}(SK_i, O_i, P_i)$ *except with negligible probability.*

To ensure the best chances that an honest relay is on the path, the honest sender will pick a path without any repetition in the relays (acyclic). Note that our adversary model trusts the sender and hence this assumption is merely a restriction of how the protocol works.
Assumption 7. *Each honestly chosen path $\mathcal{P}$ is acyclic.*

While true for, to our knowledge, all protocols, we use the following processing order as an assumption in our proofs:
Assumption 8. *Each onion is processed by the receiver, before it is replied to.*

6.3.2 Modeling Replies

We extend the definition of an onion routing scheme [31] as used in Chapter 5 with an algorithm to send replies, similar to [5].

Definition 21 (Repliable OR Scheme)**.** *A Repliable OR Scheme is a tuple of PPT algorithms* $(G, \text{FormOnion}, \text{ProcOnion}, \text{ReplyOnion})$ *defined as:*

Key generation: $G(1^\lambda, p, P_i)$ *outputs a key pair* (PK_i, SK_i) *as in Definition 14.*

Forming: $\text{FormOnion}(i, \mathcal{R}, m, \mathcal{P}^\rightarrow, \mathcal{P}^\leftarrow, (PK)_{\mathcal{P}^\rightarrow}, (PK)_{\mathcal{P}^\leftarrow})$ *returns an i-th*[1] *onion layer* O_i *(*$i = 1$ *for sending) on input of* $i \leq n + n^\leftarrow + 2$ *(for* $i > n+1$*,* m *is the reply message and* O_i *the backward onion layer), randomness* $\mathcal{R}$*, message* m*, a forward path* $\mathcal{P}^\rightarrow = (P_1, \dots, P_{n+1})$*, a backward path* $\mathcal{P}^\leftarrow = (P_1^\leftarrow, \dots, P_{n^\leftarrow+1}^\leftarrow)$*, public keys* $(PK)_{\mathcal{P}^\rightarrow} = (PK_1, \dots, PK_{n+1})$ *of the relays on the forward path, and public keys* $(PK)_{\mathcal{P}^\leftarrow} = (PK'_1, \dots, PK'_{n^\leftarrow+1})$ *of the relays on the backward path. The backward path can be empty if the onion is not intended to be repliable.*

Forwarding: $\text{ProcOnion}(SK, O, P)$ *outputs the next layer and router* (O', P') *as in Definition 14.*

Replying to an onion. $\text{ReplyOnion}(m^\leftarrow, O, P, SK)$ *returns a reply onion* $O^\leftarrow$ *along with the next router* $P^\leftarrow$ *on input of a received (forward) onion* O*, a reply message* $m^\leftarrow$*, the receiver identity* P *and its secret key* SK*.* $O^\leftarrow$ *and* $P^\leftarrow$ *attains* $\perp$ *in case of an error.*

Correctness. We want the onions to take the paths and deliver the messages that were chosen as the input to FormOnion resp. ReplyOnion.

Definition 22 (Correctness)**.** *Let* $(G, \text{FormOnion}, \text{ProcOnion}, \text{ReplyOnion})$ *be a repliable OR scheme with maximal path length* N*. Then for all* $n, n^\leftarrow < N$*,* $\lambda \in \mathbb{N}$*, all choices of the public parameter* p*, all choices of the randomness* $\mathcal{R}$*, all choices of forward and backward paths* $\mathcal{P}^\rightarrow = (P_1, \dots, P_{n+1})$ *and* $\mathcal{P}^\leftarrow = (P_1^\leftarrow, \dots, P_{n^\leftarrow+1}^\leftarrow)$*, all* $(PK_i^{(\leftarrow)}, SK_i^{(\leftarrow)})$ *generated by* $G(1^\lambda, p, P_i^{(\leftarrow)})$*, all messages* $m, m^\leftarrow$*, all possible choices of internal randomness used by* ProcOnion *and* ReplyOnion*, the following needs to hold (except with negligible probability):*

Correctness of forward path: $Q_i = P_i$*, for* $1 \leq i \leq n$ *and* $Q_1 := P_1$*,*
$O_1 \leftarrow \text{FormOnion}(1, \mathcal{R}, m, (\mathcal{P}^\rightarrow), (\mathcal{P}^\leftarrow), (PK_1, \dots, PK_{n+1}), (PK_1^\leftarrow, \dots, PK_{n^\leftarrow+1}^\leftarrow))$ *and*
$(O_{i+1}, Q_{i+1}) \leftarrow \text{ProcOnion}(SK_i, O_i, Q_i)$*.*

Correctness of request reception: $(m, \perp) = \text{ProcOnion}(SK_{n+1}, O_{n+1}, P_{n+1})$*.*

Correctness of backward path: $Q_i^\leftarrow = P_i^\leftarrow$*, for* $1 \leq i \leq n$ *and*
$(O_1^\leftarrow, Q_1^\leftarrow) \leftarrow \text{ReplyOnion}(m^\leftarrow, O_{n+1}, P_{n+1}, SK_{n+1})$*,*
$(O_{i+1}^\leftarrow, Q_{i+1}^\leftarrow) \leftarrow \text{ProcOnion}(SK_i^\leftarrow, O_i^\leftarrow, Q_i^\leftarrow)$*.*

Correctness of reply reception: $(m^\leftarrow, \perp) = \text{ProcOnion}(SK_{n^\leftarrow+1}^\leftarrow, O_{n^\leftarrow+1}^\leftarrow, P_{n^\leftarrow+1}^\leftarrow)$*.*

Recognizing onions. To define our security properties, we need a way to recognize if an onion O provided by the adversary resulted from processing a given onion O^*. To this end, we define the algorithm $\text{RecognizeOnion}(i, O, \mathcal{R}, m, \mathcal{P}^\rightarrow, \mathcal{P}^\leftarrow, (PK)_{\mathcal{P}^\rightarrow}, (PK)_{\mathcal{P}^\leftarrow})$, which uses the given inputs (that have been used to create the onion O^* in the first place) to form the i-th layer of the onion O_i^* using FormOnion and then compares the header of O_i^* to the header of the onion O in question. If the headers are identical, it returns *True*, otherwise *False*.

Note that the "correctness" of FormOnion and RecognizeOnion for $i > 1$ is defined implicitly as part of our security properties in Section 6.4.

[1]During normal operation only $i = 1$ is used. The possibility to form onion layers for $i > 1$ (without using ProcOnion) is needed for our security definitions and proofs.

6.3.3 Ideal Functionality

Similar to Chapter 5, as long as the sender is honest we want that the adversary can only learn the parts of the onion's path (and associated reply's path), where she corrupted all relays. This includes especially the following three facts:

1. The adversary cannot link onion layers before and after any honest relay.
2. The adversary cannot learn the included message, unless she controls the receiver.
3. The adversary cannot distinguish whether onions are on the forward or backward path, unless she controls the receiver and the onion is either the last layer before (forward) reception, or the first layer of her reply.

Note that this especially includes that she cannot link layers based on malleability attacks on the payload. See Appendix C.1.1 for technical details.

6.4. New Properties

We now define our security properties and show that if they are fulfilled, our ideal functionality is realized. The idea is similar to the one in Chapter 5: To ensure that the adversary only learns the subpaths from each honest relay until the next honest relay, we replace any real sequence of onion layers that is observed on such a subpath, with a random sequence that only is equal in the information learned by the adversary, i.e., the allowed leakage of the ideal functionality. For replacement, we distinguish three types of subpaths and introduce one property for each type, challenging the adversary to distinguish the real and a random layer sequence for this specific subpath type. The types are: a subpath that is part of the forward path (Forward Layer-Unlinkability), one that is part of the backward path (Backward Layer-Unlinkability) and one that includes parts of the forward and backward path as the receiver is corrupted (Repliable Tail-Indistinguishability).

Forward Path: We first require that the layers on the forward path can be replaced by *random* ones. Therefore, we extend Layer-Unlinkability from Section 5.2.4 with oracles for the creation of replies and illustrate the property in Figure 6.1.

Thereby, we challenge the adversary to distinguish between (a) an onion created according to her choices from (b) a random onion that takes the same path from the sender to the first honest relay. We use oracles to allow for processing of and replying to (other) onions at the honest relays. Due to duplicate checks, these oracles only return a processed onion if no onion with this header was processed before (Assumption 6) and only return a reply onion if the onion was processed before (Assumption 8). Further, the oracle after the challenge has to treat the challenge onion with care: if it is processed or replied to (depending if the honest relay is an intermediate relay or the receiver), an onion that fits to the original choice is constructed with FormOnion and returned.

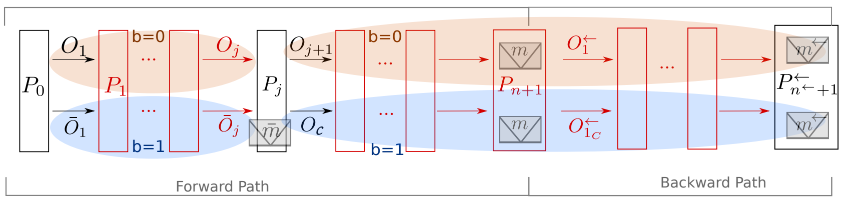


Figure 6.1: Forward Layer-Unlinkability illustrated: Red boxes are corrupted relays, black honest relays, orange ellipses are the $b = 0$ and the blue the $b = 1$ case. $\bar{m}$ is a random message. The main idea is that the adversary cannot distinguish between real and random onions *before* P_j.

Definition 23 (Forward Layer-Unlinkability $LU^{\rightarrow}$)**.** *Forwards Layer-Unlinkability is defined as:*

1. *The adversary receives the router names P_H, P_S and challenge public keys PK_S, PK_H, chosen by the challenger by letting $(PK_H, SK_H) \leftarrow G(1^\lambda, p, P_H)$ and $(PK_S, SK_S) \leftarrow G(1^\lambda, p, P_S)$.*

2. *Oracle access: The adversary may submit any number of* Proc *and* Reply *requests for P_H or P_S to the challenger. For any* Proc(P_H, O)*, the challenger checks whether η is on the η^H-list. If not, it sends the output of* $\mathrm{ProcOnion}(SK_H, O, P_H)$*, stores η on the η^H-list and O on the O^H-list. For any* Reply(P_H, O, m) *the challenger checks if O is on the O^H- list and if so, the challenger sends* $\mathrm{ReplyOnion}(m, O, P_H, SK_H)$ *to the adversary. (Similar for requests on P_S with the η^S-list).*

3. *The adversary submits a message m, a position j with $1 \leq j \leq n+1$, a path $\mathcal{P}^{\rightarrow} = (P_1, \dots, P_j, \dots, P_{n+1})$ with $P_j = P_H$,a path $\mathcal{P}^{\leftarrow} = (P_1^{\leftarrow}, \dots, P_{n^{\leftarrow}+1}^{\leftarrow} = P_S)$ and public keys for all nodes PK_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 chosen paths are acyclic, the router names are valid and that the same key is chosen if the router names are equal, and if so, sets $PK_j = PK_H$ and $PK_{n^{\leftarrow}+1}^{\leftarrow} = PK_S$ and picks $b \in \{0, 1\}$ at random.*

5. *The challenger creates the onion with the adversary's input choice and honestly chosen randomness $\mathcal{R}$: $O_1 \leftarrow \mathrm{FormOnion}(1, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}})$ and a replacement onion with the first part of the forward path $\bar{\mathcal{P}}^{\rightarrow} = (P_1, \dots, P_j)$, a random message $\bar{m} \in \mathcal{M}$, another honestly chosen randomness $\bar{\mathcal{R}}$, and an empty backward path $\bar{\mathcal{P}}^{\leftarrow} = ()$: $\bar{O}_1 \leftarrow \mathrm{FormOnion}(1, \bar{\mathcal{R}}, \bar{m}, \bar{\mathcal{P}}^{\rightarrow}, \bar{\mathcal{P}}^{\leftarrow}, (PK)_{\bar{\mathcal{P}}^{\rightarrow}}, (PK)_{\bar{\mathcal{P}}^{\leftarrow}})$*

6. *If $b = 0$, the challenger gives O_1 to the adversary.*
   Otherwise, the challenger gives $\bar{O}_1$ to the adversary.

7. *Oracle access:*

   If $b = 0$, the challenger processes all oracle requests as in step 2).

   Otherwise, the challenger processes all requests as in step 2) except for:

   – *If $j < n+1$:* Proc(P_H, O) *with* $\mathrm{RecognizeOnion}(j, O, \bar{\mathcal{R}}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}) = True$*, η is not on the η^H-list and* $\mathrm{ProcOnion}(SK_H, O, P_H) \neq \perp$*:*
   The challenger outputs (P_{j+1}, O_c) with $O_c \leftarrow \mathrm{FormOnion}(j+1, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}})$ and adds η to the η^H-list and O to the O^H-list.

   – *If $j = n+1$:*

     ∗ Proc(P_H, O) *with* $\mathrm{RecognizeOnion}(j, O, \bar{\mathcal{R}}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}) = True$*, η is not on the η^H-list and* $\mathrm{ProcOnion}(SK_H, O, P_H) \neq \perp$*:*
     The challenger outputs $(m, \perp)$ and adds η to the η^H-list and O to the O^H-list.

     ∗ Reply$(P_H, O, m^{\leftarrow})$ *with* $\mathrm{RecognizeOnion}(j, O, \bar{\mathcal{R}}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}) = True$*, O is on the O^H- list and has not been replied before and* $\mathrm{ReplyOnion}(m^{\leftarrow}, O, P_H, SK_H) \neq \perp$*:*
     The challenger outputs $(P_1^{\leftarrow}, O_c)$ with $O_c \leftarrow \mathrm{FormOnion}(j+1, \mathcal{R}, m^{\leftarrow}, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}})$

8. *The adversary produces guess b'.*

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

Note that by using the real processing for the oracle in Step 7 for $b = 0$ and the recognition and a newly formed onion layer for $j+1$ in $b = 1$, it follows that both RecognizeOnion and FormOnion have to adhere to their intuition, i.e. with overwhelming probability only the challenge onion is recognized and the newly formed layer has to be indistinguishable to the real processing.

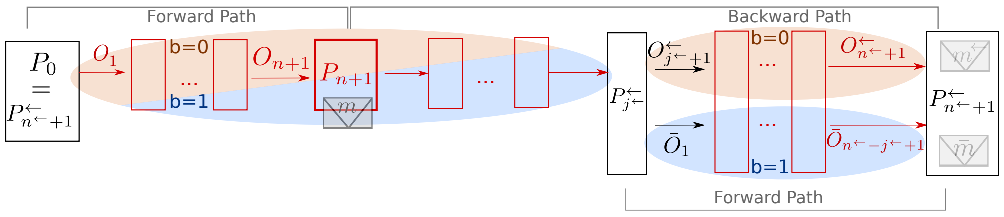

Forward Path
Backward Path
Forward Path
$P_0 = P_{n^{\leftarrow}+1}^{\leftarrow}$
O_1
b=0
b=1
O_{n+1}
P_{n+1}
m
$P_{j^{\leftarrow}}^{\leftarrow}$
$O_{j^{\leftarrow}+1}^{\leftarrow}$
$O_{n^{\leftarrow}+1}^{\leftarrow}$
$\bar{O}_1$
$\bar{O}_{n^{\leftarrow}-j^{\leftarrow}+1}$
$P_{n^{\leftarrow}+1}^{\leftarrow}$
$m^{\leftarrow}$
$\bar{m}$

Figure 6.2: Backward Layer-Unlinkability illustrated: Red boxes are corrupted relays, black honest relays, orange ellipses are the $b = 0$ and the blue the $b = 1$ case. The main idea is that the adversary cannot distinguish between real and random onions *after* $P_{j^{\leftarrow}}^{\leftarrow}$.

Backward Path: Additionally, we build a reverse version of Layer-Unlinkability for the backward path and illustrate the property *Backward Layer-Unlinkability* $LU^{\leftarrow}$ in Figure 6.2. This definition is similar to $LU^{\rightarrow}$, but the challenge is to distinguish a reply from randomness. We thus return the challenge onion in a special case of the second oracle (Step 7 in $LU^{\rightarrow}$) and the forward onion is always constructed to the adversary's choice (instead of Step 6 in $LU^{\rightarrow}$). The challenge onion either contains the layers of the reply constructed to the adversary's choices (including the chosen reply message) or random *forward* layers with a random message. As these two cases are trivially distinguishable by processing the challenge onion at the honest original sender (i.e. backwards receiver), we ensure that the oracle denies to do this final processing of the challenge onion. This corresponds to the real world, in which our trusted sender does not share any received message with the adversary. For a formal definition of this property see Appendix C.2.2.

Notice that we pick the random replacements to be *forward onion layers*. Thus the property $LU^{\leftarrow}$ implies indistinguishability between forward and backward onions for intermediates (otherwise the adversary could distinguish the real (backward) onion from the fake (forward) onion).

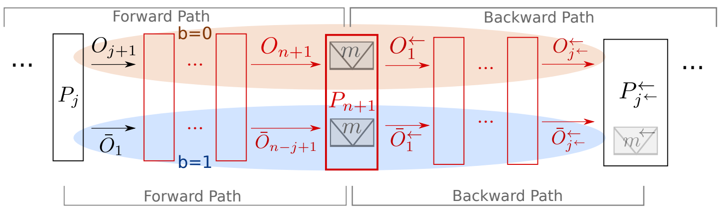


Figure 6.3: Repliable Tail-Indistinguishability illustrated: Red boxes are corrupted relays, black honest relays, orange ellipses are the $b = 0$ and the blue the $b = 1$ case. While the adversary can learn the behavior between *between* P_j and $P_{j^{\leftarrow}}^{\leftarrow}$ she cannot connect it to anything before P_j and after $P_{j^{\leftarrow}}^{\leftarrow}$.

Between forward and backward path: Finally, we want to replace the layers between the last honest relay on the forward and the first honest relay on the backward path with random ones. Note that the replaced part of the path contains an adversarial receiver. For the replacement in this case, we extend Tail-Indistinguishability from Section 5.2.3 with oracles for the creation of replies and illustrate the property *Repliable Tail-Indistinguishability* $TI^{\leftrightarrow}$ in Figure 6.3. As we can already replace all other layers before and after this part of the path with random ones due to the Layer-Unlinkability properties, the $TI^{\leftrightarrow}$ property does not output anything for these layers. We thus, start outputting the challenge layers only *after* the honest relay P_j on the forward path and refuse processing of the challenge onion at the honest relay on the backwards path $P_{j^{\leftarrow}}^{\leftarrow}$ in our oracle (similar to $LU^{\leftarrow}$). The challenge onion hence either contains the layers after P_j of an onion that is built according to the adversary's choices or random layers that take the same part of the path and carry the same message, but for an onion that actually starts at P_j and ends (with the backwards path) at $P_{j^{\leftarrow}}^{\leftarrow}$. For a formal definition of this property see Appendix C.2.2.

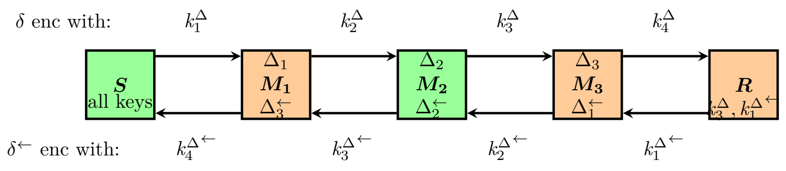


Figure 6.4: Overview of the basic idea for the payload δ. Each relay gets an updatable encryption token Δ to change the key k^Δ under which the payload is encrypted.

Properties imply ideal functionality As argued in the beginning of this section, we built the properties to step by step replace the real onion layers between honest relays with random ones that only coincide with the real ones in information that the ideal functionality allows to leak. By applying one property for each subpath between honest relays at a time, similar to earlier proofs [31] and Section 5.2.5, we show that these properties imply the ideal functionality in Appendix C.2.2. From here on, we call any OR scheme that fulfills our properties a *secure, repliable OR scheme.*

6.5. Our UE-based Scheme

We start by outlining the basic ideas of our protocol based on updatable encryption (UE) (see Fig. 6.4). The sender encrypts its request using an UE scheme and provides each relay, using a header construction similar to Sphinx, with an update token to unlinkably transform this request. Similarly, the receiver is equipped with the corresponding decryption key and a fresh encryption key for the backward path. The security properties of UE ensure that valid payloads from an honest sender cannot be modified or replaced by adversarially generated payloads as needed to protect against the malleability attack, as well as to protect the unlinkability of ciphertexts and current position in the path.

More precisely, each onion $O = (\eta, \delta)$ consists of two components:

- a *header* η which contains encrypted key material with which routers can process and (conventionally) authenticate the header itself, and
- the UE-encrypted *payload* δ; each router will use a UE update token to rerandomize and re-encrypt δ under a different (hidden) key.

The structure of η is similar to the Sphinx and Shallot protocols. Namely, each layer contains a public-key encryption (under the public key of the respective relay) of ephemeral keys that encrypt the next layer (including the address of the next relay), and authentication information with which to verify this layer. Additionally, in our case each layer also contains an encrypted token which can be used to update the payload ciphertext δ and we include the backward path in the header as well. All of this header information can be precomputed by the sender for both communication directions (i.e., for the path from sender to receiver, and for the return path).

After processing the header, each relay pads the so-extracted header for the next relay suitably with randomness, so that it is not clear how far along the onion has been processed. At some point, the decrypted header will contain a receiver symbol and a UE decryption key to indicate that processing of the onion in the forward direction has finished.

The header will then also contain a UE encryption key and enough information for the receiver to prepare a "backwards onion", i.e., an onion with the same format for the return path. We stress that all header parts of this "backwards onion", including UE tokens and authentication parts, are precomputed by the initial sender. The receiver merely UE-encrypts the payload and adds padding similarly to relays during processing.

Processing on the return path works similarly, only that eventually, the initial sender is contacted with the (still UE-encrypted) reply payload. The sender can then decrypt the payload using a precomputed UE key.

We stress that there is no explicit check that the payload δ is still intact at any point. However, the demanded UE security guarantees that re-encryption of invalid UE ciphertexts will return a failure.

6.5.1 Building Blocks

Our construction makes use of the generic building blocks listed below (see Section 2.2 for an introduction of the security requirements).

- an asymmetric CCA2 secure encryption scheme (to encrypt ephemeral keys) with encryption and decryption algorithms denoted by $\mathsf{PK.Enc}_{PK_i}$ and $\mathsf{PK.Dec}_{SK_i}$ when used with public key PK_i and secret key SK_i,
- a PRP-CCA secure symmetric encryption scheme (to encrypt routing information) of length L_1 with encryption and decryption algorithms denoted by $\mathsf{PRP.Enc}_{k^\eta}$ and $\mathsf{PRP.Dec}_{k^\eta}$ when used with the symmetric key k^η,
- an SUF-CMA secure message authentication code (to protect the header) with tag generation and verification algorithm denoted by MAC_{k^γ} and Ver_{k^γ} when used with the symmetric key k^γ,
- an UP-IND-RCCA and UP-INT-PTXT secure updatable encryption scheme with perfect re-encryption (to protect the payload) with encryption, decryption and re-encryption algorithms denoted by $\mathsf{UE.Enc}_{k^\Delta}$, $\mathsf{UE.Dec}_{k^\Delta}$ and $\mathsf{UE.ReEnc}_{\Delta}$ when used with keys k^Δ and tokens Δ. We assume that keys and tokens are of the same length or padded to the same length. Further, all messages are padded to the same length.

6.5.2 Scheme Description

Setup. To setup the system, $\mathsf{UE.GenSP}(pp)$ needs to be run on the public parameters pp (which, e.g., may contain a description of the group setting) by an honest party (or by using multi-party computation). The resulting system parameters sp need to be made public and used by all participating parties. Usually they are distributed along with the software package.

Header Construction. Each onion layer O_i, which is sent from P_{i-1} to P_i, is a tuple of header η_i and payload δ_i: $O_i = (\eta_i, \delta_i)$. Constructing the header is similar to the Sphinx approach [52] and the Shallot scheme [5]. Contrary to the existing works, we however treat the payload with sufficiently secure updatable encryption.

Each header η_i is a tuple of encrypted temporary keys and tokens in E_i, encrypted routing information and keys for the current router P_i and later routers $P_{>i}$ in B_i^j and a MAC over the header in γ_i: $\eta_i = (E_i, B_i^1, B_i^2, \dots, B_i^{2N-1}, \gamma_i)$. We describe a non-repliable header first and later on extend it to be repliable. The first layer's header η_1 contains:

$$\eta_1 = (\quad E_1, \quad B_1^1, \quad B_1^2 \quad ,\dots, \quad B_1^{2N-1}, \quad \gamma_1 \quad)$$
$$\eta_1 = (\mathsf{PK.Enc}_{PK_1}(k_1^\eta, k_1^\gamma, \Delta_1), \mathsf{PRP.Enc}_{k_1^\eta}(P_2, E_2, \gamma_2), \mathsf{PRP.Enc}_{k_1^\eta}(B_2^1), \dots, \mathsf{PRP.Enc}_{k_1^\eta}(B_2^{2N-2}), \mathsf{MAC}_{k_1^\gamma}(E_1, B_1^1, \dots, B_1^{2N-1}))$$

The second layer's header η_2 has padding added by the first relay in B_2^{2N-1}:

$$\eta_2 = (\quad E_2, \quad B_2^1, \quad B_2^2 \quad ,\dots, \quad B_2^{2N-1}, \quad \gamma_2 \quad)$$
$$\eta_2 = (\mathsf{PK.Enc}_{PK_2}(k_2^\eta, k_2^\gamma, \Delta_2), \mathsf{PRP.Enc}_{k_2^\eta}(P_3, E_3, \gamma_3), \mathsf{PRP.Enc}_{k_2^\eta}(B_3^1), \dots, \mathsf{PRP.Dec}_{k_1^\eta}(\mathbf{0}\dots\mathbf{0}), \mathsf{MAC}_{k_2^\gamma}(E_2, B_2^1, \dots, B_2^{2N-1}))$$

The already existing relay padding is further decrypted for later layers:

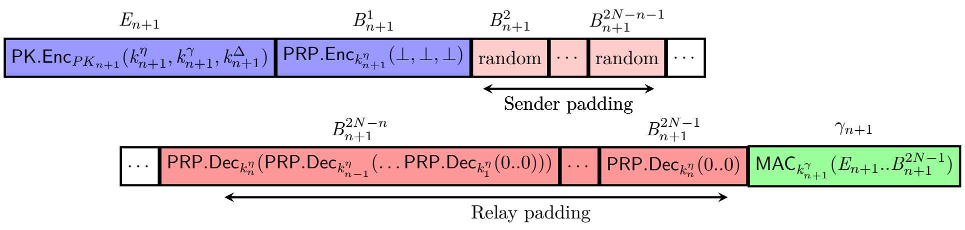

E_{n+1}
B^1_{n+1}
B^2_{n+1}
B^{2N-n-1}_{n+1}
$\mathsf{PK.Enc}_{PK_{n+1}}(k^\eta_{n+1}, k^\gamma_{n+1}, k^\Delta_{n+1})$
$\mathsf{PRP.Enc}_{k^\eta_{n+1}}(\perp,\perp,\perp)$
random
$\cdots$
random
$\cdots$
Sender padding
B^{2N-n}_{n+1}
B^{2N-1}_{n+1}
γ_{n+1}
$\cdots$
$\mathsf{PRP.Dec}_{k^\eta_n}(\mathsf{PRP.Dec}_{k^\eta_{n-1}}(\ldots\mathsf{PRP.Dec}_{k^\eta_1}(0..0)))$
$\cdots$
$\mathsf{PRP.Dec}_{k^\eta_n}(0..0)$
$\mathsf{MAC}_{k^\gamma_{n+1}}(E_{n+1}..B^{2N-1}_{n+1})$
Relay padding

Figure 6.5: Non-repliable receiver header illustrated

$$\eta_3 = (\ldots, \quad B_3^{2N-3} \quad , \quad B_3^{2N-2} \quad , \quad B_3^{2N-1} \quad ,\ldots)$$
$$\eta_3 = (\ldots, \mathsf{PRP.Enc}_{k_3^\eta}(B_4^{2N-4}), \mathsf{PRP.Dec}_{k_2^\eta}(\mathsf{PRP.Dec}_{k_1^\eta}(\mathbf{0}\ldots\mathbf{0})), \mathsf{PRP.Dec}_{k_2^\eta}(0\ldots0),\ldots)$$

The message is destined for the current processing relay P_{n+1} if $(\perp,\perp,\perp)$ is encrypted in B^1_{n+1}. All later $B^{>1}_{n+1}$ contain random bit strings chosen by the sender resp. the padding added by the earlier relays (see Figure 6.5). The blocks with sender chosen padding are used for the reply path in repliable onions later.

To construct η_1 for a path $\mathcal{P} = (P_1, \ldots, P_{n+1}), n+1 \le N-1$, the sender builds the onion from the center, i.e. calculates the layer for the receiver first:

1. Pick keys $k_1^\eta, \ldots, k_{n+1}^\eta$ for the block cipher, $k_1^\Delta, \Delta_1, \ldots, \Delta_n, k_{n+1}^\Delta$ for the UE and $k_1^\gamma, \ldots, k_{n+1}^\gamma$ for the MAC randomly.

2. Construct η_{n+1}:

$$\begin{aligned}
E_{n+1} &= \mathsf{PK.Enc}_{PK_{n+1}}(k^\eta_{n+1}, k^\gamma_{n+1}, k^\Delta_{n+1})\\
B^1_{n+1} &= \mathsf{PRP.Enc}_{k^\eta_{n+1}}(\perp,\perp,\perp)\\
B^{2N-i}_{n+1} &= \mathsf{PRP.Dec}_{k^\eta_n}(\mathsf{PRP.Dec}_{k^\eta_{n-1}}(\ldots\mathsf{PRP.Dec}_{k^\eta_{n+1-i}}(0\ldots0)))\\
&\quad \text{for } 1 \le i \le n \text{ (blocks appended by relays)}\\
B^{2N-i}_{n+1} &\leftarrow^R \{0,1\}^{L_1} \text{ for } n+1 \le i \le 2N-2\\
&\quad \text{(blocks as path length padding calculated by sender)}\\
\gamma_{n+1} &= \mathsf{MAC}_{k^\gamma_{n+1}}(E_{n+1}, B^1_{n+1}, B^2_{n+1}, \ldots, B^{2N-1}_{n+1})
\end{aligned}$$

3. Construct $\eta_i, i < n+1$ recursively (from $i = n$ to $i = 1$):

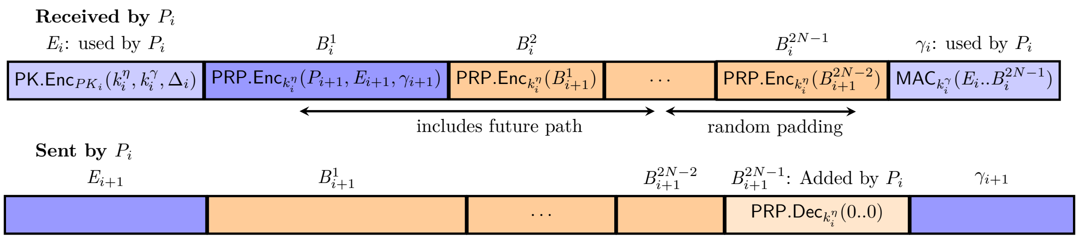
$$\begin{aligned}
E_i &= \mathsf{PK.Enc}_{PK_i}(k_i^\eta, k_i^\gamma, \Delta_i)\\
B_i^1 &= \mathsf{PRP.Enc}_{k_i^\eta}(P_{i+1}, E_{i+1}, \gamma_{i+1})\\
B_i^j &= \mathsf{PRP.Enc}_{k_i^\eta}(B_{i+1}^{j-1}) \text{ for } 2 \le i \le 2N-1\\
\gamma_i &= \mathsf{MAC}_{k_i^\gamma}(E_i, B_i^1, B_i^2, \ldots, B_i^{2N-1})
\end{aligned}$$

Payload Construction. Let m be a message of the fixed message length to be sent. We add a 0 bit to the message to signal that it is not repliable $m' = 0\|m$:

$$\delta_i = \mathsf{UE.ReEnc}_{\Delta_{i-1}}(\ldots(\mathsf{UE.ReEnc}_{\Delta_1}(\mathsf{UE.Enc}_{k_1^\Delta}(m'))\ldots).$$

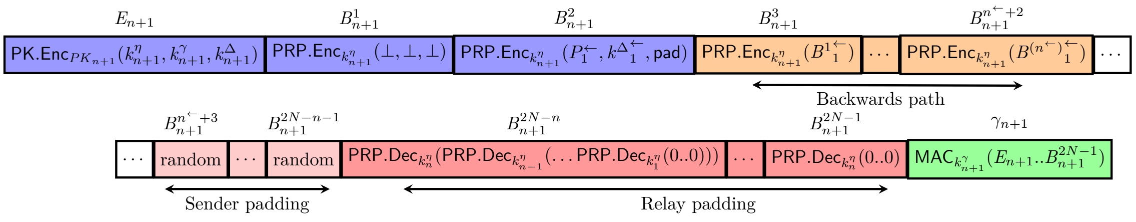
Onion Processing. The same processing is used for any forward or backward, repliable or not-repliable onion. If P_i receives an onion $O_i = (\eta_i = (E_i, B_i^1, B_i^2, \ldots, B_i^{2N-1}, \gamma_i), \delta_i))$, it takes the following steps (see Fig. 6.6):

1. Decrypt the first part of the header $(k_i^\eta, k_i^\gamma, \Delta_i) = \mathsf{PK.Dec}_{PK_i}(E_i)$ [resp. k^Δ_{n+1} instead of Δ_i, if P_i is the receiver]



Received by P_i
E_i: used by P_i
B_i^1
B_i^2
B_i^{2N-1}
γ_i: used by P_i
$\mathsf{PK.Enc}_{PK_i}(k_i^\eta, k_i^\gamma, \Delta_i)$
$\mathsf{PRP.Enc}_{k_i^\eta}(P_{i+1}, E_{i+1}, \gamma_{i+1})$
$\mathsf{PRP.Enc}_{k_i^\eta}(B_{i+1}^1)$
$\dots$
$\mathsf{PRP.Enc}_{k_i^\eta}(B_{i+1}^{2N-2})$
$\mathsf{MAC}_{k_i^\gamma}(E_i..B_i^{2N-1})$
includes future path
random padding
Sent by P_i
E_{i+1}
B_{i+1}^1
B_{i+1}^{2N-2}
B_{i+1}^{2N-1}: Added by P_i
γ_{i+1}
$\dots$
$\mathsf{PRP.Dec}_{k_i^\eta}(0..0)$

Figure 6.6: Processing illustrated



E_{n+1}
B_{n+1}^1
B_{n+1}^2
B_{n+1}^3
$B_{n+1}^{n^\leftarrow+2}$
$\mathsf{PK.Enc}_{PK_{n+1}}(k_{n+1}^\eta, k_{n+1}^\gamma, k_{n+1}^\Delta)$
$\mathsf{PRP.Enc}_{k_{n+1}^\eta}(\perp, \perp, \perp)$
$\mathsf{PRP.Enc}_{k_{n+1}^\eta}(P_1^\leftarrow, k^{\Delta}{}_1^\leftarrow, \mathsf{pad})$
$\mathsf{PRP.Enc}_{k_{n+1}^\eta}(B^1{}_1^\leftarrow)$
$\dots$
$\mathsf{PRP.Enc}_{k_{n+1}^\eta}(B^{(n^\leftarrow)}{}_1^\leftarrow)$
Backwards path
$B_{n+1}^{n^\leftarrow+3}$
B_{n+1}^{2N-n-1}
B_{n+1}^{2N-n}
B_{n+1}^{2N-1}
γ_{n+1}
random
random
$\mathsf{PRP.Dec}_{k_n^\eta}(\mathsf{PRP.Dec}_{k_{n-1}^\eta}(\dots\mathsf{PRP.Dec}_{k_1^\eta}(0..0)))$
$\mathsf{PRP.Dec}_{k_n^\eta}(0..0)$
$\mathsf{MAC}_{k_{n+1}^\gamma}(E_{n+1}..B_{n+1}^{2N-1})$
Sender padding
Relay padding

Figure 6.7: Repliable receiver header illustrated

2. Check the MAC γ_i of the received onion (and abort if it fails)

3. Decrypt the second part of the header $(P_{i+1}, E_{i+1}, \gamma_{i+1}) = \mathsf{PRP.Dec}_{k_i^\eta}(B_i^1)$ [if $P_{i+1} = E_{i+1} = \gamma_{i+1} = \perp$ (P_i is the receiver), skip processing of the header and process the payload (and check for replies as explained below)]

4. Decrypt the rest of the header $B_{i+1}^{j-1} = \mathsf{PRP.Dec}_{k_i^\eta}(B_i^j)$ for $j \geq 2$

5. Pad the new header $B_{i+1}^{2N-1} = \mathsf{PRP.Dec}_{k_i^\eta}(0\dots0)$

6. Construct the new payload $\delta_{i+1} = \mathsf{UE.ReEnc}_{\Delta_i}(\delta_i)$ [resp. retrieve the message in case of being the receiver ($\delta_{i+1} = \mathsf{UE.Dec}_{k^\Delta_{n+1}} = 0\|m$ if no reply)] and abort if this fails

7. Send the new onion $O_{i+1} = ((E_{i+1}, B_{i+1}^1, \dots, B_{i+1}^{2N-1}, \gamma_{i+1}), \delta_{i+1})$ to the next relay P_{i+1}

Constructing a Repliable Onion. Let m be the message for the receiver, $\mathcal{P}^\leftarrow = (P_1^\leftarrow, \dots, P_{n^\leftarrow+1}^\leftarrow)$, $n^\leftarrow + 1 \leq N - 1$ the backward path. To send a repliable onion, the sender performs the following steps:

1. Construct a (non-repliable) header $\eta_1^\leftarrow$ with path $\mathcal{P}^\leftarrow$. Let the chosen keys be $k^\eta{}_1^\leftarrow, \dots, k^\eta{}_{n^\leftarrow+1}^\leftarrow$ and $k^\Delta{}_1^\leftarrow, \Delta_1^\leftarrow, \dots, \Delta_{n^\leftarrow}^\leftarrow, k^\Delta{}_{n^\leftarrow+1}^\leftarrow$

2. Construct the (repliable) header η_1 by starting to construct η_{n+1} for the receiver as before in the non-repliable case, but with the following differences (see Fig. 6.7) where pad is the padding to the fixed blocklength:
   - Set $B_{n+1}^2 = \mathsf{PRP.Enc}_{k_{n+1}^\eta}(P_1^\leftarrow, k^\Delta{}_1^\leftarrow, \mathsf{pad})$
   - Set $B_{n+1}^i = \mathsf{PRP.Enc}_{k_{n+1}^\eta}(B^{(i-2)}{}_1^\leftarrow)$ for $3 \leq i \leq n^\leftarrow + 2$
   - Store the key $k^\Delta{}_{n^\leftarrow+1}^\leftarrow$

3. Evolve the header η_{n+1} as before to create η_1

4. Construct the message for the repliable onion as $m' = 1\|m$

5. Construct the payload δ_1 for m' as before

6. The repliable onion is (η_1, δ_1)

Sending a reply. After recognizing to be the receiver (due to $(\perp, \perp, \perp)$ in B^1) of an repliable message (due to the starting bit), the receiver retrieves $P_1^{\leftarrow}$ and $k^{\Delta}{}_1^{\leftarrow}$ from B_{n+1}^2. Let $m^{\leftarrow}$ be the reply message padded to the fixed message length. To send the reply the receiver performs the following steps:

1. Calculate $\delta_1^{\leftarrow} = \mathsf{UE.Enc}_{k^{\Delta}{}_1^{\leftarrow}}(m^{\leftarrow})$
2. Evolve the header (as before but shifting the header by two blocks):
   - $B^{(j-2)}{}_1^{\leftarrow} = \mathsf{PRP.Dec}_{k_{n+1}^{\eta}}(B_{n+1}^{j})$ for $j \geq 3$
   - $B^{(2N-1)}{}_1^{\leftarrow} = \mathsf{PRP.Dec}_{k_{n+1}^{\eta}}(0 \dots 0)$ (i.e. receiver padding)
   - $B^{(2N-2)}{}_1^{\leftarrow} = \mathsf{PRP.Dec}_{k_{n+1}^{\eta}}(1 \dots 1)$ (i.e. receiver padding)
3. Send the onion $O_1^{\leftarrow} = (\eta_1^{\leftarrow}, \delta_1^{\leftarrow})$ to $P_1^{\leftarrow}$

Decrypting a reply. After recognizing to be the receiver (due to $(\perp, \perp, \perp)$ in B^1), the relay checks whether the included key k_{n+1}^{Δ} for her matches a stored $k^{\Delta}{}_{n^{\leftarrow}+1}^{\leftarrow}$s (it indeed is a reply) or not (it is just a new message). She uses the key and decrypts the message: $m = \mathsf{UE.Dec}_{k_{n+1}^{\Delta}}(\delta)$.

6.6. Security of Our Repliable OR Scheme

In this section, we prove that our scheme is secure (see Appendix C.4.1 for a formal definition of the used security requirements):
Theorem 8. *Assume a CCA2 secure PKE, a PRP-CCA secure SKE, a UP-IND-RCCA, and UP-INT-PTXT secure UE scheme with perfect re-encryption (of arbitrary ciphertexts), and a SUF-CMA secure MAC are given. Then our construction described in Section 6.5 satisfies* $LU^{\rightarrow}$ *security.*

Intuitively, the CCA2 secure PKE ensures that the temporary keys for each relay are only learned by the intended relay, and the PRP-CCA secure SKE that the header is rerandomized and can be padded in the processing at a relay (so incoming and outgoing onions cannot be linked based on the header). Further, the SUF-CMA secure MAC protects the header against modifications. The UE scheme takes care of the payload: the UP-IND-RCCA ensures that the message is hidden and that the payload is rerandomized during the processing at a relay (so incoming and outgoing onions cannot be linked based on the payload), UP-INT-PTXT security that the payload cannot be maliciously modified (as in the malleability attack), while Perfect Re-Encryption guarantees that the adversary does not learn how far on the path the onion has already traveled.

Formally, we first describe FormOnion for later layers and show a detailed proof sketch for $LU^{\rightarrow}$. As the proofs for $LU^{\leftarrow}$ and $TI^{\leftrightarrow}$ are similar to the one of $LU^{\rightarrow}$, we only quickly sketch them here. All detailed proofs are provided in Appendix C.4.2. Further, correctness follows from inspection of our scheme.

FormOnion for $i > 1$. FormOnion for $i > 1$ uses the k_i^{Δ} belonging to the corresponding epoch to create the payload $\delta = \mathsf{UE.Enc}_{k_i^{\Delta}}(m)$ and creates the other onion parts deterministically as described in the protocol for the current layer (with the randomness, all used keys are known and the deterministic parts of all layers can be built). For reply layers ($i > n + 1$) it combines the deterministically computed header and payload with the encrypted new message (as all randomness is known, all temporary keys are).

Forwards Layer-Unlinkability. Our proof for $LU^{\rightarrow}$ follows a standard hybrid argument. We distinguish the cases that the honest node is a forward relay ($j < n+1$) and that it is the receiver ($j = n+1$).

Case 1 – Honest Relay ($j < n+1$). We first replace the temporary keys of the honest party included in the header, to be able to change the blocks of the header and the payload corresponding to the $b = 1$ case. For the oracles we further need to ensure, that RecognizeOnion does not mistreat any processing of e.g. modified onions. Therefore, we leverage the UE properties for the payload protection and the MAC for the header. Table C.2 in Appendix C.4.2 provides an overview of the proof sketch. The full proof that can be found in [98] is rather extensive and hence omitted from this thesis.

Proof. [Proof sketch] We assume a fixed, but arbitrary PPT algorithm $\mathcal{A}_{LU^{\rightarrow}}$ as adversary against the $LU^{\rightarrow}$ game and use a sequence of hybrid games $\mathcal{H}$ for our proof. We show that the probability of $\mathcal{A}_{LU^{\rightarrow}}$ outputting $b' = 1$ in the first and last hybrid are negligibly close to each other.

Hybrid 1) $LU^{\rightarrow}_{(b=0)}$. The $LU^{\rightarrow}$ game with b chosen as 0.

Hybrid 2) replaces the keys and token included in E_j with $0 \ldots 0$ before encrypting them and adapts the oracle of Step 7 such that RecognizeOnion checks for the adapted header, but still uses the original keys as decryption of E_j.

We reduce this to the CCA2 security of our PK encryption: We either embed $0 \ldots 0$ or the keys and token as the $CCA2$ challenge message and process other onions (for the Step 7 oracle) by using the $CCA2$ decryption oracle.

Hybrid 3) rejects all onions that reuse E_j, but differ in another part of the header, in the oracle of Step 7.

Due to the SUF-CMA of our MAC a successful processing of a modified header can only occur with negligible probability.

Hybrid 4) replaces the blocks (with information and keys for the future path of the onion) with random blocks and adapts the oracle of Step 7 such that RecognizeOnion checks for the adapted header, but still uses the original blocks as processing result.

We reduce this to the PRP-CCA security of the PRP, by embedding the PRP-CCA challenge into these blocks, while continuing to treat these same blocks during processing as if they had the original content. (Other blocks in onions using E_j are rejected in the oracle of Step 7.)

Hybrid 5) replies with a fail to all Step 7 oracle requests, that use the challenge onion's header, but modified the message included in its payload.

We reduce this to the UP-INT-PTXT of our UE: First, we carefully construct the secrets of the challenge onion until it is at the honest relay with the help of the UP-INT-PTXT-oracles. Then we wait for an onion with the challenge header to be given to the oracle in Step 7. We use the payload of this onion as the ciphertext to break UP-INT-PTXT. Note that we do not have to answer this oracle request in our reduction, but only oracle requests for onions with a different header, which we can easily process with the knowledge of the secret keys (only the keys for the challenge onion are partially unknown in the reduction).

Hybrid 6) replaces the processing result of the challenge onion (recognized based on the header, with an unchanged message in payload) with a newly formed onion (FormOnion) that includes the same rest of the path and message.

FormOnion constructs the header deterministically as before, the only difference is the re-encryption (Hybrid 5) and the fresh encryption (Hybrid 6) of the same message in the payload. Due to the perfect Re-Encryption of our UE scheme those are indistinguishable.

Hybrid 7) replaces the message included in the payload with a random message and adapts the oracle in Step 7 to expect this random message as payload, but still replies with the newly formed onion including the original message as before.

We reduce this to the UP-IND-RCCA security of our UE: We carefully construct the secrets of the challenge onion until the honest relay with the help of the UP-IND-RCCA-oracles and either embed the original or a random message as the UP-IND-RCCA challenge message. To answer the Step 7 oracle, we use the knowledge of the secret keys if the requested onion does not have the challenge onion's header. If it has, we use the decryption oracle of UP-IND-RCCA to detect whether the payload was maliciously modified (the UP-IND-RCCA oracle returns another message m') or not (the UP-IND-RCCA oracle does not process the payload). In the first case, we return a fail (as introduced in Hybrid 5), in the second we return a newly formed onion (as introduced in Hybrid 6).

Hybrid 8) - Hybrid 12) revert the Hybrids 5)-2) (similar argumentation), which results finally in he $LU^{\rightarrow}$ game with b chosen as 1. $\square$

Case 2 – Honest Receiver ($j = n + 1$): We sketch the proof in Table C.3 of Appendix C.4.2 and provide more details in [98]. The steps are the same as for the first case of $LU^{\rightarrow}$, but in Hybrid 6) we need to treat Reply and Proc requests separately. As the FormOnion behavior simulating the receiver is exactly the same as in the real protocol, we do not need to rely on Perfect Re-Encryption, but just on correctness of the decryption in this step. Note further that the earlier restrictions on the oracle work both for Reply and Proc requests.

Other Properties. We sketch the proofs in Table C.4 – C.6 of Appendix C.4.2 and provide more details in [98].

Theorem 9. *Assume a CCA2 secure PKE, a PRP-CCA secure SKE and a UP-IND-RCCA secure UE scheme with perfect Re-Encryption (of arbitrary ciphertexts), and a SUF-CMA secure MAC are given. Then our construction described in Section 6.5 satisfies $LU^{\leftarrow}$ security.*

Backwards Layer-Unlinkability. The steps are similar to the ones for $LU^{\rightarrow}$ Case 1: We replace the temporary keys of honest routers, before we exclude bad events (header manipulations) at the oracles and finally set the header and payload parts to correspond to the $b = 1$ case. However, this time we need to replace parts for both at the forward and backward path, as the forward layers also include information about the backward layers (but not the other way round). Notice that we can skip the steps related to the modification of the payload (and thus UP-INT-PTXT). Because the forward message is known to the adversary anyways and the backward message (as the final processing) is never given to the adversary, she cannot exploit payload modification at the oracles to break $LU^{\leftarrow}$.

Theorem 10. *Assume a CCA2 secure PKE, a PRP-CCA secure SKE, and a SUF-CMA secure MAC are given. Then our construction described in Section 6.5 satisfies $TI^{\leftrightarrow}$ security.*

Tail-Indistinguishability. This is similar to $LU^{\leftarrow}$, except that we can skip more steps. For the same reasons as before, we do not need the payload protection in $TI^{\leftrightarrow}$. Further, the adversary does not obtain any leakage related to k_j^{η} and thus the blocks in the forward header can be replaced right away.

6.7. Our SNARG-based Scheme

We now present an alternative instantiation of a secure, repliable OR scheme based on SNARGs, instead of updatable encryption. Our SNARG-based protocol works conceptually similarly, but with two differences:

- First, the payload is enclosed by multiple symmetric encryption layers (one for each relay). This is very similar to previous approaches [52, 5], but also opens the door to malleability attacks.

- Second, in order to prevent such malleability attacks, each layer contains a concise SNARG proof on top of header and payload, which proves that this onion is the result of (a) a fresh onion as constructed by a sender, (b) a fresh backwards onion as constructed by a receiver, or (c) a legitimate processing of another onion (with a valid SNARG proof). In essence, this SNARG proof avoids malleability attacks by inductively proving that this onion has gone only through valid onion generation or processing steps.

We note that the SNARG proof may need to show that this onion is the result of an honest processing of another onion *with a valid SNARG proof.* Hence, we need to be careful in designing the corresponding SNARG language in a recursive way while avoiding circularities. This recursive and self-referential nature of our language is also the reason why we use SNARGs (as opposed to "regular" zero-knowledge techniques with larger proofs).

Our crucial tool to enable this recursive "reverse-processing" is the soundness of the used SNARG. Each onion thus will carry enough encrypted information to "recreate" previous onions, and the corresponding SNARG will certify the validity of that (encrypted) information. Since the size of onions should not grow during processing, we will not be able to *fully* reconstruct the previous onion. However, reconstructing parts of it suffices for our strategy. Further, like before, we rely on using MACs for a more "fine-grained" (and, most importantly, deterministic) authentication and progression of onion *headers.*

Viewed from a higher level, these consistency proofs provide a whole authentication chain for both requests and replies even with an intermediate receiver that replies with an arbitrary (and a-priori unknown) payload. This authentication chain protects against malleability attacks and payload changes along the way.

6.7.1 Building Blocks and Setting

Our construction makes use of the generic building blocks listed below (see Section 2.2 for an introduction of the security requirements) and emphasize the differences compared to the UE-based scheme.

- an asymmetric CCA2-secure encryption scheme with encryption and decryption algorithms denoted by $\mathsf{PK.Enc}_{PK_i}$ and $\mathsf{PK.Dec}_{SK_i}$ when used with public key PK_i and secret key SK_i.

- an SUF-CMA secure message authentication code with tag generation algorithm denoted by MAC_{k^γ} when used with the symmetric key k^γ.

- *two* PRP-CCA secure symmetric encryption schemes of short length L_1 (for the header) and long length L_2 (for the payload) with encryption and decryption algorithms denoted by $\mathsf{PRP.Enc}_{k^\eta}$ and $\mathsf{PRP.Dec}_{k^\eta}$ resp. $\mathsf{PRP2.Enc}_{k^\delta}$ and $\mathsf{PRP2.Dec}_{k^\delta}$ when used with the symmetric key k^η resp. k^δ.

- a *rerandomizable CPA-secure asymmetric encryption scheme*, with en-, decryption and rerandomization algorithms denoted by $\mathsf{PKM.Enc}_{PK^{\mathrm{M}}}$, $\mathsf{PKM.Dec}_{SK^{\mathrm{M}}}$ and $\mathsf{PKM.ReRand}_{PK^{\mathrm{M}}}$ when used with public key PK^{M} and secret key SK^{M}. We require that rerandomization is invertible, in the sense that knowing the random coins of $\mathsf{PKM.ReRand}$ allows to retrieve the original ciphertext.

- a *simulation-sound SNARG* with proof generation, verification, and simulation algorithms denoted by $\mathsf{Prove}_{\mathsf{ZK}}$, $\mathsf{Vfy}_{\mathsf{ZK}}$, and $\mathsf{Sim}_{\mathsf{ZK}}$.

We assume that all keys of honest participants are chosen independently at random. Regarding the setting, we assume additionally that a master public key PK^{M} (for the rerandomizable CPA-secure encryption) and a common reference string CRS (for the SNARG) are known to all participants, while the corresponding SNARG trapdoor and secret key SK^{M} are not known to anyone.[2]

We will use PK^{M} to let participants encrypt secrets "to the sky", and the corresponding secret key SK^{M} will only be used as an extraction trapdoor in our proof. Hence, it is crucial that in an implementation of our scheme, both PK^{M} and CRS are chosen such that noone knows their trapdoors. (However, at least in the case of CRS, subversion-zero-knowledge SNARKs [68] are a promising tool to allow for adversarially chosen CRS.)

6.7.2 Scheme Description

In our protocol, each onion $O = (\eta, \sigma, \delta)$ consists of three main components:

- a *header* η which contains encrypted key material with which routers can process and (conventionally) authenticate the onion header,
- the (SNARG-related) *authentication part* σ,
- the multiply encrypted *payload* δ; each router will decrypt one layer during processing.

While η and δ are similar to the Sphinx and Shallot protocols, σ contains several SNARG proofs $\pi_1, \dots, \pi_N$ and an encrypted ring buffer (that consists of ciphertexts $C_1, \dots, C_N$). Here, N denotes the maximal path length in the scheme. Intuitively, the C_i contain information that is required to reverse-process O, and the π_i prove that the information encrypted in C_1 is accurate. More specifically:

- C_1 contains a public-key encryption of the $\pi'_1, \dots, \pi'_N$ from the *previous* onion O', as well as the last router's long-term secret key SK'. The public key used is a public parameter of the OR scheme, such that the secret key is not known by anyone. Of course, this last property is crucial to the security of the scheme. We will use this secret key as a trapdoor that allows to reverse-process onions during the proof.
- $C_2, \dots, C_N$ are the values $C'_1, \dots, C'_{N-1}$ from O'. Note that this implies that C'_N is lost during processing and cannot be reconstructed.
- π_i is a SNARG proof that proves that η, δ, and $C_1, \dots, C_{N-i}$ are the result of an honest processing of some previous onion. The reason for N proofs π_i (and not just a single one) is that during repeated reverse-processing of a given onion, more and more C_i will unavoidably be lost. To check the integrity of such incomplete onions, we will use π_i in the i-th reverse-processing step.

Header Construction. Each onion layer O_i is a tuple of header η_i, SNARG-Information σ_i and payload δ_i: $O_i = (\eta_i, \sigma_i, \delta_i)$. We construct the header η_i as in the UE-based solution (see Section 6.5.2), except that instead of the Δ_i resp. k_i^{Δ} we now include k_i^{δ} of the second PRP-CCA secure symmetric encryption scheme for the relays.

SNARG Construction. The SNARG-Information σ_i consists of a ring buffer $C_i = (C_i^1, \dots, C_i^N)$ and the SNARGs $\pi_i = (\pi_i^1, \dots \pi_i^N)$: $\sigma_i = (C_i, \pi_i)$.

Ring buffer. The ring buffer C_i is calculated similarly to B_i, but reversed. The ring buffer for forward onions includes all information needed to undo the processing of the onion or reconstruct all input to FormOnion, encrypted under the master public key. On the reply path, we

[2]Those public parameters can be either chosen by a trusted party, agreed upon with an initial multi-party computation, or, if SNARG and the rerandomizable encryption scheme have dense keys, be derived from a public source of trusted randomness (like, e.g., sunspots).

overwrite old (forward) information in C_is, as this is sufficient to achieve the forward-backward indistinguishability.

$$\begin{aligned}
C_1^1 &= \mathsf{PKM.Enc}_{PK^{\mathrm{M}}}(\mathcal{I}) \text{ with } \mathcal{I} = (\texttt{form}, (R, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}))\\
C_1^j &\leftarrow^R \{0,1\}^{L_3} \setminus \{\texttt{sim}\} \text{ with } \texttt{sim} \text{ being a special symbol}\\
&\qquad \text{and } L_3 \text{ the fixed length of ring buffer elements}\\
C_i^1 &= \mathsf{PKM.Enc}_{PK^{\mathrm{M}}}(\mathcal{I}) \text{ with } \mathcal{I} = (\texttt{proc}, (SK_{i-1}, \pi_{i-1}^1, \dots, \pi_{i-1}^N, E_{i-1}, P_{i-1}))\\
C_i^j &= \mathsf{PKM.ReRand}_{PK^{\mathrm{M}}}(C_{i-1}^{j-1})
\end{aligned}$$

Note that the onion O_i is created by P_{i-1} and thus the information included in C_i is known at the time of creation. Further, information encrypted in C_i does not include the payload message or the MAC, as both can be reconstructed given the current onion layer. Finally, all C_i^j are padded to the fixed length L_3.

SNARGs. The SNARG π_i^j is calculated by P_{i-1} for the language $\mathcal{L}^j$, which consists of all partial onions $X = (\eta_i, (C_i^1, \dots, C_i^{N-j}), \delta_i)$ for which the following holds: namely, there should exist R, M such that $C_i^1 = \mathsf{Enc}(PK^{\mathrm{M}}, M; R)$, and such that M fulfills the following:

1. If M is of the form $M = (\texttt{form}, I)$, then I is some parameter list $I = (1, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}})$ (including random coins $\mathcal{R}$) for which FormOnion(I) outputs an onion $O^* = (\eta^*, \sigma^*, \delta^*)$ with $\eta^* = \eta_i$ and $\delta^* = \delta_i$. In other words, in this case, M explains X as an immediate FormOnion output for a particular message m.
2. If M is of the form $M = (\texttt{proc}, (SK_{i-1}, \pi_{i-1}^1, \dots, \pi_{i-1}^N, E_{i-1}, P_{i-1}, \mathcal{R}))$, then
   (a) all π_{i-1}^{N-k} (for $k > j$) are valid, in the sense that π_{i-1}^{N-k} shows that $(\eta_i, (C_i^1, \dots, C_i^{N-k}), \delta_i) \in \mathcal{L}^k$. (Note that this is a well-defined statement if we define $\mathcal{L}^j$ for larger j first.)
   (b) $\text{ProcOnion}_{\text{partial}}^j(SK_{i-1}, (\eta_{i-1}, (C_{i-1}^1, \dots, C_{i-1}^{N-j-1}), \delta_{i-1}), P_{i-1}; \mathcal{R}) = (\eta_i, (C_i^1, \dots, C_i^{N-j}), \delta_i)$, where $\text{ProcOnion}_{\text{partial}}^j$ is the upcoming ProcOnion algorithm restricted to header, payload, and (partial) ring buffer processing (i.e., without any SNARG proof checks or creations), and η_{i-1}, δ_{i-1}, and the C_{i-1}^j are the previous header, payload, and (partial) ring buffer that are reverse-processed from X, SK_{i-1}, and random coins $\mathcal{R}$.[3]
3. M of any other form are not allowed.

The intuition behind $\mathcal{L}^j$ is simple: partial onions in $\mathcal{L}^j$ feature a ciphertext C_i^1 that allows to "reverse-process" the given onion to some extent. In particular, either the onion in question is the immediate output of either a FormOnion or a ProcOnion query. In case of a ProcOnion output, the whole onion cannot be reconstructed or checked (since some information in the C_i ring buffer is necessarily lost during processing). However, given an onion O_i and the secret key SK^{M}, the validity of π_i^N guarantees that a large portion of O_{i-1} can be reconstructed. In fact, only C_{i-1}^N cannot possibly be retrieved. However, going further, the reconstructed π_{i-1}^{N-1} now makes a statement about that "incomplete onion" O_{i-1}, and the reverse-processing can be continued.

Payload Construction. For message m, we again signal that it is not repliable by prepending a 0-bit: $m' = 0 \| m$ and construct the payload as multiple encryption:

$$\delta_1 = \mathsf{PRP2.Enc}_{k_1^\delta}(\mathsf{PRP2.Enc}_{k_2^\delta}(\dots \mathsf{PRP2.Enc}_{k_{n+1}^\delta}(m') \dots))$$

Onion Processing. The processing of the header is done as in the UE-based scheme (see Section 6.5.2). However, the processing also checks the SNARG and treats the payload with

[3]We will describe ProcOnion only below, but it will be clear that the header, payload, and partial ring buffer part of the processing can be reversed with the secret key SK_{i-1} of the processing party. We additionally run $\text{ProcOnion}_{\text{partial}}$ to re-check MAC values.

$\mathsf{PRP2.Dec}$:

If P_i receives an onion $O_i = (\eta_i = (E_i, B_i^1, B_i^2, \dots, B_i^{2N-1}, \gamma_i), (C_i, \pi_i), \delta_i))$, it does the following steps differently:

1. Check the SNARG-Sequence π_i of the received onion (and abort if it fails)
2. Decrypt the header to retrieve $(k_i^\eta, k_i^\gamma, k_i^\delta)$ and new header blocks for $i+1$, check the MAC, pad the header as before (see Section 6.5.2)
3. Construct the new payload $\delta_{i+1} = \mathsf{PRP2.Dec}_{k_i^\delta}(\delta_i)$ [resp. retrieve the message in case of being the receiver ($\delta_{i+1} = 0\|m$ if no reply)]
4. Rerandomize and shift ring buffer: $C_{i+1}^{j+1} = \mathsf{PKM.ReRand}_{PK^{\mathrm{M}}}(C_i^j)$
5. Replace first ring buffer entry: $C_{i+1}^1 = \mathsf{PKM.Enc}_{PK^{\mathrm{M}}}(\mathcal{I})$
6. Build the new SNARG-Sequence π_{i+1}
7. Send the new onion $O_{i+1} = ((E_{i+1}, B_{i+1}^1, \dots, B_{i+1}^{2N-1}, \gamma_{i+1}), (C_{i+1}, \pi_{i+1}), \delta_{i+1})$ to the next relay P_{i+1}

Constructing a Repliable Onion. The construction of a repliable onion works as for the UE-based scheme before, except that we include $k^{\delta}{}_1^{\leftarrow}$ in the header and store all chosen $k^{\delta}{}_i^{\leftarrow}$ for later use.

Sending a reply. Processing the repliable onion, the receiver stores $P_1^{\leftarrow}, \eta_1^{\leftarrow}$ and k_R^δ. To reply with m (padded to the fixed message length), the receiver does the following steps:

1. Calculate $\delta_1 = \mathsf{PRP2.Enc}_{k_R^\delta}(m)$
2. Construct the SNARG-Sequence π_1
3. Pick the ring buffer elements randomly $C_1^j \leftarrow^R \{0,1\}^{L_3} \setminus \{\mathtt{sim}\}$ for all j
4. Send the onion $O_1 = (\eta_1^{\leftarrow}, (C_1, \pi_1), \delta_1)$ to $P_1^{\leftarrow}$

Decrypting a reply. After recognizing to have received a reply (by checking the stored $k_{n^{\leftarrow}+1}^\delta$), the reply is "decrypted":

$$m = \mathsf{PRP2.Dec}_{k_1^\delta}(\mathsf{PRP2.Enc}_{k_2^\delta}(\dots(\mathsf{PRP2.Enc}_{k_{n^{\leftarrow}}^\delta}(\mathsf{PRP2.Enc}_{k_{n^{\leftarrow}+1}^\delta}(\delta))\dots)))$$

6.7.3 Security

The proofs of our onion routing properties are similar to the ones for the UE-based scheme, except that they rely on the SNARGs to protect the payload. We detail them in Appendix C.4.3.
Theorem 11. *Our SNARG-based OR Scheme is a* secure, repliable OR scheme.

7. Beyond Anonymous Communication

Besides the privacy requirements on the higher layers of anonymous communication (e.g., network or application layer), on the lower technological transmission layers, namely the physical layer, other notions of information-theoretic secure design are considered [25]. With the help of our notions from Chapter 3, we were able to provide the first equivalence results between privacy notions in anonymous communication and security definitions on the physical layer (see [106] for details).

Further, we investigated fundamental privacy goals and existing solutions for two other applications. We detail the corresponding first results on proximity tracing and payments below.

7.1. Proximity Tracing Notions

Motivated by the importance of automatic contact tracing due to the current SARS-CoV-2 pandemic, we decided to aid the design of privacy-preserving solutions with our knowledge about privacy notion definitions. We thus define important notions for proximity tracing applications in this section and study the proposed solutions regarding our new definitions in Appendix D.1.3. Results of this section have been published at the Elsevier Journal on Online Social Networks and Media 2021 [96]. We note that the results, especially regarding the proposed solutions originally stem from April 2020 and hence might not apply for newer versions of the solutions.

7.1.1 Background

Functionality. In traditional contact tracing patients with a positive test are inquired about the people they have met throughout the characteristic contagion period of the respective disease, and subsequently each of the identified individuals is informed to self-quarantine for the incubation time. This approach is highly privacy invasive, yet very specific: the tested individual is requested to release all contact information to the health authorities (or an organizing institution to which such services are delegated), surrenders the set of encountered individuals that still come to mind, and that the person feels comfortable to disclose. The institution thus gets to know these individuals and quite accurate social interactions.

Extreme situations, like the current SARS-CoV-2 pandemic, require alternative approaches. We observe that as incubation periods are long, it is difficult for tested individuals to remember all encounters. Further as large numbers of people are tested positive, manual interviewing and calling of encountered individuals does not scale sufficiently.

Several suggestions have been made to use location information from cell phones for this purpose. They fundamentally comprise of three functions: i) sensing potential infection events, either by tracking all users' locations, or encounters between groups of users, ii) deriving "at-risk" (potentially infected) users upon the event of one user being tested positive, and iii) informing the at-risk users to take further measures (e.g., self-isolate, get tested). Note, that at least one bit of information is transferred to the alerted user, that is, the information that he or she has met at least one individual who has tested positive for the virus. This can fundamentally only be avoided by non-deterministic protocols that allow for false negatives and false positive alarms. The epidemiologists, however, require deterministic protocols to ensure efficient use of test kits and avoiding to miss potential infections.

They commonly assume different underlying *roles and participating stakeholders.* The most basic assumption is that individuals are participating as the owners of their mobile phones (we will call their representation within the app "users"); individuals may be untested, tested positive, or, after an encounter with a positive individual, "at-risk". Testing at least in Europe is assumed to be performed by *medical professionals.* The results of positive tests are collected at public *health authorities*, who also take care of informing at-risk individuals.

Privacy Threats. A contact tracing system processes the (co-)location of all of its users (in some form) and the information who has been tested positive for the respective virus. Both are considered especially sensitive by all relevant data protection laws, as they relate to health and potentially religious and political opinions. Processing these types of information with perfect knowledge about all users clearly is privacy invasive.

Different approaches hence try to prevent certain threats: These comprise leaking *infection of an individual* and *potential infection of an individual*, for instance of all those who are alerted for being at risk. Also leaking *infection* or *potential infection of a pseudonymous user* is a threat, as inverting a pseudonym and re-identifying the individual may be possible, sometimes even easy, especially for long-lived pseudonyms, through different means. Additional threats are leaking the *location of an individual*, even the *location of a pseudonymous user* (as this may allow to learn the whereabouts of another individual, to track a user, and, as previous work based on pseudonymous location information has shown, to simple re-identification [56]), or *encounters of two users* (as personal relationships are intimate information, and even highly distorted social graph information has been shown to be easily re-identified [121]). Even leaking aggregate information could represent a threat, as stigmatization may happen if, for example, *frequencies of infections*, or *fractions of infected populations* become known for users with certain behavior, or in certain regions, cities, or probably even specific shops, companies, or institutions.

Adversary and Trust Assumptions. With regard to the adversaries, the common view is that the systems should protect an individual's information from the *health authorities* (or their delegated service providers), *other users*, or *third parties* who abuse the system for attacks on the privacy of its users, for instance by placing devices (Bluetooth antennas, video cameras, corrupted smart phones) in certain locations. Further attacks on the system itself, like, for instance *pollution* (illegitimate infection information, claiming a positive test outcome), or *replay/DoS* (re-broadcasting previously received tokens so as to increase the probability of this user to be alerted for being at risk) are imaginable. *We acknowledge the importance of attacks on the system, but limit our scope on attacks on the privacy of users within this thesis.*

The approaches have entirely different *trust assumptions*: For *centralized* systems, the service is assumed to be trusted and so it is perfectly fine for the service to learn all encounters of individuals with those who are infected, but the information of who has been infected is hidden from all users. Even those who are alerted that they are considered to be at risk do not learn which or how many infected individuals they have met, unless the alerted only met a single person. *Decentralized* approaches take the opposite stance and hide location traces and colocations from the service, but in the extreme case allow the alerted users to learn who of their contacts has been tested positive if the users store the information which token has been broadcast during encounters with which other individual. Further storing the location for each broadcast token even allows alerted users to recover the location of each individual that tests positive (at the time of encounters). *Hybrid* forms between the two extremes and *distributed* trust between different entities of the protocols are further variations.

Missing Formal Privacy Definitions. *No formal privacy definitions and security analyses of the suggested approaches have been provided at the time of this writing*, to the best of our knowledge. Cho, Ippolito, and Yu [46] have performed an informal analysis of the TraceTogether App that is in use in Singapore, and suggest some improvements to prevent specific attacks. Vaudenay [155] has reported some specific vulnerabilities of a version of the DP3T protocol. Neither of the two, nor any of the protocol descriptions that we could retrieve has made an attempt to define the potential risk formally with actual security goals, to make the systems accessible to formal analysis.

In brief, we identify a dire necessity to formalize the setting, trust assumptions, and accurate privacy notions in order to allow for a systematic understanding and discussion of the different proposals that are circulated, and their flavor of privacy which they may, or may not achieve.

7.1.2 Application Scenario and Assumptions

Humans get infected and potentially transmit diseases. We consider the humans to be *individuals*, with names (e.g. 'Alice' and 'Bob'). In possession of a cell phone they can become *users* with corresponding pseudonyms, some of which can be (easily) linked back to the individual (the pseudonym hence *inverted* or *mapped*) - like for instance contact information (the mobile number, or the app identifier). For some pseudonyms this may not be simple (short term random IDs), or probably even impossible for others. We assume that users are able to learn their own pseudonyms (based on technical expertise and physical control over their own devices).

Functionality. At-risk individuals, i.e. those who have been in the proximity of another individual that has tested positive in the disease specific time span, have to be alerted of their situation. We assume the notification has to happen at the earliest possible time.

We make the assumption that infected individuals safely self-quarantine and that their positive test is not projected forward throughout quarantine. Otherwise, enabled alerts at infected users could potentially be used for hypothesis testing (passing by somebody's home address and checking if this results in an alert). Further, after the quarantine they are treated just as new users are treated with a new user ID. This assumption allows for less complex definitions. Note that while practical solutions do not change the user ID explicitly after infection, they ensure this kind of limited leakage by only disclosing changing pseudonyms instead of a user ID.

Roles. We consider three entities to be involved: *the individuals* who install and use the app (and who may or may not become sick or at-risk), *the service*, who can be the responsible health authorities, a delegated service provider or a combination of providers, and potential additional *third parties*, who are running one or several external devices. Entities in any of these roles might try to break the privacy of selected, any, or all users of the application and have different adversarial capabilities, like being strictly passively observing or actively deviating from the protocol.

The effectiveness of a proximity tracing system depends directly on its adoption, and we hence assume open groups without access limitations, so it is easy for anybody to participate. Even though it is out of scope for this work, we want to note that this does not necessarily entail zero-cost identities and hence unbounded Sybil attacks, as linking, for instance, installation identifiers to mobile devices, SIM cards, or phone numbers is possible, and sufficiently secure at least against opportunistic attacks.

Data. The service needs to consider colocation, which translates to two individuals being at the same location at the same moment. We hence have to model locations and time.

We do not model *location* in terms of the coordinates, or traces of location where users have been. Several analyses (e.g. [56]) have shown that anonymizing location data, especially when sequences of positions are known, is very difficult (and most likely impossible, if any utility is expected to remain). Privacy-preserving processing or publishing of such data hence comes at least at a great overhead. Instead, we model only colocation as the fact that two users have been within a defined proximity of each other, no matter where this encounter has taken place. Colocation data is strictly less rich in information as it only reveals that two people met, but nothing about the corresponding location (unless the adversary has external information). This decision conforms with most of the discussed protocols (all contestants for implementation as European solutions).

Regarding the *time*, we need to distinguish between two different intervals: Batch time as a basic abstraction of time for our formal game model and retention time as a disease specific time window.

- Batch time: Each batch groups seemingly simultaneous events. They are so short (e.g. seconds) that an adversary cannot recognize the sequence of events in them.
- Retention time (e.g. 2 weeks): The amount of time the proximity information is needed.

Data Processing and Information Loss. We deem personal information of users, and potentially aggregates thereof, sensitive and aim to analyze to which extent it is kept private (or, otherwise: lost).

The service processes data, and both functionality and service provision require transfer of certain information. Like any relevant privacy-preserving protocol, our service must leak at least a minimum of information, as otherwise it would have no utility: Potentially infected "at-risk" users learn at least the single bit of information that they are at risk, and hence have encountered one or some individuals that later have tested positive. In addition the server, and all current protocol proposals include some centralized servers to facilitate communication, will learn at least the fact that it is used, so that users exist. We do not deem service information (the existence, effort, cost, etc. of the service provision) worth protecting.

We also limit our analysis to the information that leaks *due to using the service*, so that an adversary learns from using the application, or controlling the service or parts of it, or combinations thereof. For now, we only study the conceptual solutions proposed, but do not consider any other channels, e.g. leaks due to underlying techniques like Bluetooth. We however stress that our privacy definitions can directly be used in such an extended analysis, it is just out of scope for our protocol investigation in this work.

We do not analyze the secrecy of external (probably private) information, as it can easily be collected by other means (side channels like setting up cameras in public places, walking around town and taking notes of observations, digital dossier aggregation). It is important to note that we make no assumptions nor statements about such external information, and, as argued by Dwork [59], that even the smallest loss of information in combination with external knowledge can constitute an absolute privacy loss. Thus, absolute privacy guarantees can of course not be derived from our study (and most probably not from any other).

Adversary. The adversary might participate in any role. It can be a curious user, the service provider (if the protocol design includes one), or external third parties. Our formalization of privacy goals can also be used to analyze colluding adversaries, that corrupt multiple users, or a user *and* third parties to learn private information. Due to the assumption of open groups, an adversarial service provider can also easily create artificial users in collusion and thereby merge the observations at the service provider and the controlled users.

7.1.3 Model

We model events, which the adversary can choose to define the scenarios, and a notion of time that fits for this purpose. We borrow the general idea of game based notions combined with ideas of differential privacy from anonymous communication networks as in Section 3.2.

Events. We use $\mathcal{U}$ as the fixed set of individuals who use the proximity tracing app, and further need to model two events:

Colocation of individuals (Meeting) $m(A, B)$ meaning that individual A was in proximity of individual B. We assume this event to be symmetric. Hence, $m(B, A)$ describes the same event.

Infection of an individual $i(A)$ meaning that A was tested positive. As we assume that such users get a new user-ID for the time after the infection (and at the time they self-isolate), the same infection event is not allowed to occur multiple times.

Events that follow from these, like creating pseudonyms as user identifiers, and notifying other users to be tested or self-quarantine (e.g. B above), will follow from the app protocol and are not needed for defining the scenarios. (They will be, of course, important for investigating protocols regarding the notions, later on).

Time. We need a sense of time. Therefore we assume events to be clustered into batches $\underline{r} = (r_1, \ldots, r_l)$ with $r_i \in \{m(\cdot,\cdot), i(\cdot), \diamond\}$, where $\diamond$ is the empty event, i.e. no event taking place, which can be used to model passing of time when no event happens. All events of a batch are assumed to happen simultaneously (or at least with so little time difference that the adversary cannot observe their order). Every new batch contains all events of this very short time span. The next batch then contains the events of the following time interval of the same length. Multiple identical colocation events (e.g. $m(A, B)$ and $m(B, A)$) are not allowed in the same batch as one cannot meet someone multiple times at the same point in time. Further, the information is stored (and compared) for a chosen retention time; conforming to t batches.

Non-modeled attributes. If needed for the protocol analysis, we assume that attributes we did not explicitly model are equal in both scenarios and are chosen reasonably, i.e. such that the chosen events might happen in the real world. For example the location at which a corrupted user meets someone in the batch is fixed and the location of her meeting in the next batch should be reasonably close. Similarly, if time is used by the protocol, it is equal in both scenarios for the same batch index.

Game. We use the general game, extensions and advantage definition as introduced in Section 3.2. Recall especially that the adversary can corrupt users with the help of corruption queries in the game.

Notions. Our notions define what the adversary is not allowed to learn. The scenarios, which the adversary has to distinguish in the game, hence have to be identical except for exactly the piece of information that the corresponding notion requires to be protected. The challenger checks for this in its check for validity, in Step 3 of the game. As before, we describe the allowed differences in the input batches as a combination of properties. These properties usually require that certain data or metadata of the real world has to be equal in both scenarios (i.e. it is not protected by the notion and can be learned by the adversary).

Further notation. Let C_i be the set of corrupted users in the i-th batch, as defined by the user corruption queries above. We use C as a shorthand for all C_i's. Note that by definition of the corruption query $C_i \subseteq C_{i+1}$. We use $[x, y]$ short for $\{x, x+1, \ldots, y\}$ and $\underline{r}[x, y]$ as an abbreviation for all events happening in the batches $\underline{r}^x, \underline{r}^{x+1}, \ldots, \underline{r}^y$. We also use the underscore "_" to declare that this part of a tuple can be of arbitrary value.

7.1.4 Properties

In this section we describe the properties, i.e. partial information about what happens in the real world or is represented in the protocol. This information may, or may not be desirable to

Table 7.1: Informal Summary of the Properties

Notation	Name	Meaning
E_i	Equal except infected	Protects all information about infected, nothing else
E_m	Equal except meetings	Protects all information about meetings, nothing else
$\lvert U_i \rvert$	Number of infected	The number of infected users is not protected
$\lvert M \rvert$	Number of meetings	The number of meeting events happening is not protected
M_i	Meetings of infected	Infected peoples' meetings are not protected
M_c	Meetings of corrupted	Corrupted peoples' meetings are not protected (thus identity of meeting partner might be leaked).
UL	Unavoidable loss	Information assumed to be always learned by the adversary

hide from an adversary. Hiding a certain property is a characteristic of a specific protocol. The property otherwise (most probably) can be observed by the adversary as an artifact of the protocol execution. There certainly are numerous possible properties that theoretically could be considered. We restrict ourselves to those properties that we find to be most insightful, and which relate well to the current protocol proposals (Appendix D.1.2 further elaborates the situation).

Informal

Informally speaking, we want a property that helps us to express that *no* information leaks, other than what is unavoidable (necessary for the functionality). Therefore, we define the *unavoidable loss* (UL) that per se has to be identical in both scenarios, as it cannot be protected under the given requirements for the solution.

Further, we want to be able to analyze the protection regarding the information loss about the colocations/meetings and the infected patients. Therefore, we define a property (E_m) that allows us to easily require that no information about the colocations, but possibly any information about the infected users (except for their colocations with other users) is leaked. We require for it that in both scenarios exactly the same infection events happen, but leave complete freedom of choice to the adversary regarding the meeting events.

Conversely, we define another property (E_i) that protects only the information of infected users and fixes the colocation events.

Achieving strong notions of course is favorable, as any information leakage may be reason for concern, especially when considering auxiliary information from external sources. Some design decisions (local broadcasting of tokens, disclosing the number of encounters to the server, or through the server to the public, etc), however, may be useful from a pragmatic perspective, yet systematically prevent protocols from achieving such strong notions against different adversaries. We thus introduce weaker notions and restrict these properties slightly by requiring further restrictions: for the infected protection we require that the number of infected users is equal in both scenarios (as this usually is published anyways); for the colocation protection that the colocation events of infected users might leak (as some protocols leak more information about the infected users) or that the colocation events of corrupted users might leak (as corrupted users learn the pseudonyms of their meeting partners and can link them to other information in some protocols). We summarize the intuition of our properties in Table 7.1.

Formal Definition

We now define the properties. For each a protocol designer can decide whether this information should be protected or not.

Decision space. For all following properties we assume that the adversary already sent $k-1$ batches per scenario and for these the challenger already checked that each pair was compliant to the notion analyzed. Now the next pair of batches was sent by the adversary and we define under which conditions they are valid. The current batches are thus the k-th batches of the game: $\underline{r}_0^k = (r_{01}, \dots, r_{0l})$, $\underline{r}_1^k = (r_{11}, \dots, r_{1l})$.

Definition 24 (Properties)**.** *The current batches fulfill the following properties if:*
Equal except infected E_i***:*** *Any event, that is not an infection event is equal in both batches.*

$$\text{For } b \in \{0,1\}\text{: } r_{bj} = m(\_,\_) \implies r_{1j} = r_{0j}, \text{ for all } j \in \{1,\dots,l\}$$

Equal except meetings E_m***:*** *Any event, that is not a meeting event is equal in both batches.*

$$\text{For } b \in \{0,1\}\text{: } r_{bj} = i(\_) \implies r_{1j} = r_{0j}, \text{ for all } j \in \{1,\dots,l\}$$

Number of infected is equal $|U_i|$***:*** *The number of infected users is equal in both scenarios.*

$$|U_0| = |U_1|, \text{ with } U_b = \{u \mid i(u) \in \underline{r}_b^k\}$$

Number of meetings is equal M***:*** *The total number of meeting events is equal in both scenarios.*

$$|M_0| = |M_1|, \text{ with } M_b = \{r \mid r = m(\_,\_) \in \underline{r}_b^k\}$$

Meetings of infected are equal M_i***:*** *With this restriction, no one can be infected if their meetings in the last t batches differ in the two scenarios. Therefore, we define the set of users whose meetings in the two scenarios differ at some point in the last t batches (i.e. at least one user is involved in the meeting of one scenario that is not in the other) as the set $U_{\overline{m}}$. This is the set of all users u that were involved in a meeting (that was event j' of batch j) which differs in at least one user in the two scenarios.*

$$\begin{aligned} U_{\overline{m}} :=& \{u \mid \exists j \in [k-t+1, k], \exists j' : \\ & m(u_1,u_2) = \underline{r}_{0j'}^j \wedge m(u_3,u_4) = \underline{r}_{1j'}^j \wedge (u_1 \neq u_3 \vee u_2 \neq u_4) \wedge (u \in \{u_1,u_2,u_3,u_4\})\} \end{aligned}$$

We define M_i to be true iff for the set of infected user U_b (U_b defined as in Number of infected $|U_i|$):

$$U_{\overline{m}} \cap U_b = \emptyset.$$

Meetings of corrupted are equal M_c***:*** *With this restriction, no one can be corrupted if their meetings in the last t batches differ in the two scenarios. $U_{\overline{m}}$ defined as above. We define M_c to be true for the current set of corrupted users C_k iff*

$$U_{\overline{m}} \cap C_k = \emptyset.$$

Unavoidable information loss. We define properties that cannot be hidden from the adversary as they belong to the intended functionality or we consider it inherent in the technical solution that is assumed. This is a delicate choice, as it implicitly – and inaccurately – accepts that some information cannot be hidden (we will comment on these decisions, below).

At-risk users always learn at least one bit due to the function of the service (that they met some (one or more) infected users). Consider, for example, a user that met only one other user and then is warned. This trivially discloses that the individual she met is infected. More generally, also if more users are met: Unless one limits the functionality (e.g. by allowing false-positives, false-negatives or delaying the warning) an adversary can always trivially distinguish the two scenarios, if her choice leads to notification (or lack thereof) for different corrupted users in the two cases. To prevent our game adversary to break our notions based on this trivial attack type, we add the property $M^{I \to C}$ and require all notified, corrupted users to be equal in both scenarios.

Definition 25. ***At risk corrupted users are equal*** $M^{I \to C}$***:*** *We want that the same corrupted users get a notification for being at risk in both scenarios. Therefore, we define the set of users that both get notified and are corrupt, to be equal. The users that get notified are those that have, in the last t batches, been in contact with any infected user u_i (whose infection is detected).*

$$N_b := \{u \in \mathcal{U} \mid \exists u_i \in \mathcal{U} : i(u_i) \in \underline{r}_b^k \wedge (m(u, u_i) \in \underline{r}_b{}^{[k-t+1,k]} \vee m(u_i, u) \in \underline{r}_b{}^{[k-t+1,k]})\}$$

Thus requiring the subset of the corrupted notified users to be equal is:

$$N_0^C = N_1^C, \text{ where } N_b^C := N_b \cap C_k.$$

If a corrupted user meets another corrupted user, she also knows her identity ($M^{C \to C}$). So we restrict how corrupted users meet each other:

Definition 26. ***Corrupted meeting the same corrupted*** $M^{C \to C}$***:*** *The same corrupted users have been meeting with each other in the last t batches.*

$$N_0^{C'} = N_1^{C'}, \text{ where}$$

$$N_b^{C'} := \{(c_1, c_2) \mid c_1, c_2 \in C_j : (m(c_1, c_2) \in \underline{r}_b{}^{[k-t+1,k]} \vee m(c_2, c_1) \in \underline{r}_b{}^{[k-t+1,k]})\}$$

Further, if a corrupted user is tested positive, the adversary learns this information ($U_i^{C \to}$). So we restrict the infection of corrupted users:

Definition 27. ***Infection events of corrupted users*** $U_i^{C \to}$***:*** *Every corrupted user is infected at the same time in both scenarios.*

$$U_{i0}^C = U_{i1}^C, \text{ where}$$
$$U_{ib}^C := \{u \in C \mid i(u) \in \underline{r}_b{}^k\}$$

Additionally, if we assume broadcasting of temporal identifiers[1] any user learns how many encounters they had. We thus define $Q_m^{C \to}$ such that the adversary cannot trivially break the notions based on this knowledge:

Definition 28. ***Frequency in meeting for corrupted users*** $Q_m^{C \to}$***:*** *Every corrupted user had the same number of colocation events in the last t batches in both scenarios. Therefore, we define the set of encounters for a user u as:*

$$M_u := \{j \in [1, l], i \in [k-t+1, k] \mid \exists u' : m(u, u') \vee m(u', u) = \underline{r}_{bj}{}^i\}$$

That the amount of encounters per user has to be the equal for the corrupted users in the last t batches is thus:

$$Q_0^C = Q_1^C, \text{ where } Q_b^C := \{(u, n) \mid u \in C_k, n = |M_u|\}$$

If we assume broadcasting of temporal identifiers also the fact whether multiple corrupted users met the same user at the same time, is always leaked:

Definition 29. ***Meeting with corrupted*** M_{c-c}***:*** *If multiple corrupted users met the same user in one scenario, they also all meet one user in the other scenario (although this user might be a different one with the same health status). Let U_0 be the infected users of the first and U_1 of the second scenario (as in the definition of the number of infected). For all corrupted users $c, c' \in C_k$ meeting an infected user $u \in U_0$, event $j, j' \in [1, l]$ it holds that:*

$$\begin{aligned} &r_{0j} \in \{m(c, u), m(u, c)\} \wedge r_{0j'} \in \{m(c', u), m(u, c')\} \\ &\quad \implies \exists u' \in U_1 : r_{1j} \in \{m(c, u'), m(u', c)\} \wedge r_{1j'} \in \{m(c', u'), m(u', c')\} \end{aligned}$$

[1]Note that other assumptions about the technical solution lead to adapted unavoidable leakage.

Table 7.2: Proximity Tracing Notion Definition

Name	Properties
Proximity Tracing Indistinguishability ($P-IND$)	UL^C
Infection Indistinguishability ($I-IND$)	$E_i \wedge UL^C$
Infected Indistinguishability ($I \setminus \lvert I\rvert-IND$)	$E_i \wedge \lvert U_i\rvert \wedge UL^C$
Meeting Indistinguishability ($M-IND$)	$E_m \wedge UL^C$
Remote Colocation Indistinguishability ($R-IND$)	$E_m \wedge \lvert M\rvert \wedge M_C \wedge UL^C$
Remote Healthy Colocation Indistinguishability ($R\setminus I-IND$)	$E_m \wedge \lvert M\rvert \wedge M_C \wedge M_i \wedge UL^C$

Similar for all corrupted users meeting a healthy user.

The combination of all properties in 7.1.4 (the notified corrupted users, the meetings between corrupted users, the detection of infections from corrupted user, the frequency in meetings for corrupted users, and meeting with corrupted) is trivially available to the adversary. We hence define the combination to be unavoidable loss:

Definition 30. ***Unavoidable loss*** UL^C

$$UL^C := M^{I\to C} \wedge M^{C\to C} \wedge U_i^{C\to} \wedge Q_m^{C\to} \wedge M_{c-c}$$

7.1.5 Notions

Any combination of the above or potentially other, additional properties (like e.g. from Appendix D.1.2), either fixed or defined by the adversary, constitutes a notion in principle. We refrain from enumerating all and instead focus on the combinations that relate to claims or seemingly achieved protection in the most prominent proposals that have been circulated, as they seem most relevant. We define them in Table 7.2 and illustrate all introduced notions with a short example for valid scenario choices in Appendix D.1.1.

Proximity Tracing Indistinguishability ($P-IND$). The adversary only learns what she trivially already knows or has to learn, but nothing else. This is the strongest notion, given that the assumptions on the functionality and technological limits hold.

Infection Indistinguishability ($I-IND$). This is the strongest notion only protecting infections. The batches only differ in who is infected and we exclude the trivial attacks as above. Notice that this also protects e.g. how many users are infected, how often someone was colocated with an infected individual and how many infected individuals someone has encountered. Thus all users, that had contact to all notified corrupted users define the anonymity set. In case of many corrupted users working together and immediate notification after detection of an infection, this set might still be small in reality.

Infected Indistinguishability ($I \setminus |I|-IND$). This notion is a little bit weaker than $I-IND$ as it additionally allows the number of infected app users to be learned by the adversary. As infection numbers are usually published (albeit cumulated and after some delay), we do not consider this privacy loss as very critical.

Meeting Indistinguishability ($M-IND$). This is the strongest notion that only protects meetings. The batches can only differ in who met, but not who is infected. This notion does not define protection of who is infected, but who someone met, and in consequence where someone has been.

Healthy meeting Indistinguishability ($M \setminus I-IND$). We now relax the above. This notion only protects the meetings of healthy people, the identity or location of infected individuals may be disclosed. As soon as someone is infected, her whole meeting history is allowed to leak. This means only users whose meetings are identical before can be infected (as stated earlier we assume that infected users self-isolate and have no later meetings).

Remote Colocation Indistinguishability (R−IND). Except for meetings with corrupted users and the number of edges in the colocation graph, no information about it leaks. Note that this notion can be easily adapted to protect the number of edges in the colocation graph by removing $|M|$.

Remote Healthy Colocation Indistinguishability (R\I−IND). Except for meetings with corrupted users and the number of edges in the colocation graph, no information about the colocation graph between non infected users leaks.

Hierarchy

$$\begin{tikzcd}
 & & P{-}IND \arrow[dl] \arrow[dr] & \\
 & M{-}IND \arrow[dl] \arrow[dr] & & I{-}IND \arrow[d] \\
M\backslash I{-}IND \arrow[dr] & & R{-}IND \arrow[dl] & I \setminus |I|{-}IND \\
 & R\backslash I{-}IND & &
\end{tikzcd}$$

Figure 7.1: Hierarchy of defined proximity tracing privacy notions

By definition of the notions, and the corresponding limitations in the valid adversarial choices, the hierarchy as presented in Figure 7.1 follows.

This hierarchy especially means that if $P{-}IND$ is achieved, any other defined notion is achieved as well; and if $R\backslash I{-}IND$ and $I \setminus |I|{-}IND$ are both not achieved, none of the defined notions is achieved. Further, the notions protecting all meeting information $M{-}IND$ and $M\backslash I{-}IND$ also imply the notions protecting the respective social graph ($R{-}IND$ and $R\backslash I{-}IND$) and are hence stronger and better connected to each other than to the notions concerned with differing infection events ($I{-}IND$ and $I \setminus |I|{-}IND$). However, there might be useful weaker notions that are both implied by notions concerned with meeting information *and* by notions concerned with infection information, e.g. how many meetings an infected user had earlier.

7.1.6 Application

In Appendix D.1.3, we relate proposed solutions to our notions. Our high-level approach already shows trade-offs between trust in different entities of the application and the connected privacy leakage. With complete trust in the server (e.g. the central location data collecting tracing server) it is relatively easy to ensure our strongest privacy goal against adversarial clients, even if several of them collude. Limiting the information the server learns about colocations (the NTK/ROBERT approach [125] / [131]), multiple clients might infer some information about other users' meetings[2], while the server still learns enough to break nearly all our notions. The only information protected, and hence notions achieved, relate to the encounters between healthy, benign users. By moving the calculation of a warning to the clients (Canetti [34] approach, DP3T [152]) we think it is possible to achieve most of our notions against a corrupted server, at the cost of even a single client breaking some of our strong notions. Hybrid protocols between centralized and decentralized approaches, like DESIRE [35] and ConTraCorona [22], offer a trade-off between those two worlds.

We stress, however, that our study of proposed solutions is just a first high-level investigation of the used general techniques at the time of writing (April 2020) and does not include any formal proofs. Moreover, we only considered simple adversary models (either the server or the clients, but no combinations) to gain first insights into the trade-offs. Practically, of course besides more in-depth analysis of the achieved protection, further criteria, like the performance, cost and development time of the system, need to be considered to decide on the best proposal in an urgent situation.

[2]The information inferred by multiple clients in the NTK/ROBERT approach is in our current analysis considered unavoidable leakage.

7.2. Payment Networks

Experience with Bitcoin, once cherished as the privacy-preserving alternative to other digital payment methods, has demonstrated that it effectively only provides very weak pseudonymity for transactions [145, 112, 135, 116, 38, 91, 132]. Privacy-Preserving Payment (P3) systems have thus become the focus of interest, as the fundamental requirement for anonymous payments has not lost its relevance. Most P3 systems however define their privacy goals ad-hoc, tailored to a research idea, or even entirely informal, despite the growing interest in the field.

Without precise and comparable definitions, the research community is missing out on a common ground to understand and tackle the complex problem of provable privacy for payments. Similar to the case in ACNs, we thus need an analysis framework to compare the privacy that is offered by different P3 systems. While some academic efforts have already surveyed the P3 solutions space for blockchains and cryptocurrencies [73, 48, 20, 3, 157], we identify a clear lack of rigorous, formal comparisons of the privacy that the systems can guarantee. To the best of our knowledge, Amarasinghe, Boyen, and McKague [3] are the first to consider the P3 problem with some rigor. However, they primarily focused on adversary models. Which detailed information is actually protected, which arguably is of at least the same significance, remained outside of their scope. To propel P3 systems and facilitate provable privacy, there is indeed a dire need for comparable payment notions that define the fine-grained variations for the protection-worthy information during transactions.

We thus adapt our results for ACNs from Chapter 3 to P3 systems to provide a foundation for this use case as well. Note however that ACNs while they can be employed to improve the privacy of P3 systems indirectly, differ from P3 systems at the conceptual level. Firstly, ACNs can solve confidentiality with the help of well-known encryption, while the payment system inherently has to consider confidentiality on payment values. Secondly, unlike ACNs, payment systems may not require sending messages and payment ledger correctness can be ensured using commitments to the payment values and ZK proofs. Indeed, barring a few solutions such as [137, 122], most privacy solutions are not immediately effective in both ACNs and P3 domains.

In this section, we contribute to the investigation of the differences as we define and compare privacy goals for P3 systems following our idea of indistinguishability games with properties as basic building blocks. We further shed light on the practically resulting anonymity set sizes that correspond to a selection of our notions. For the last goal, we conduct a simulation study. We first generate various sets of payment data using a model that is based on measured assumptions, and later employ a real world data set. The results allow us to review the effect of typical design decisions, like leaking the value of payments or deciding on delays, on the effective anonymity set size and its potential reduction. Further, we identify potential adaptations: we observe effects of changing system parameters, like increasing latency or payment frequency. We also provide evidence for improvements by simple strategies: scaled value buckets, for instance, prove useful to considerably increase the anonymity sets, at the cost of paying at most 10% more than the intended value. Additionally, we relate existing solutions to our new payment privacy notions and simulation results in Appendix D.2.3.

7.2.1 Background

Payment Systems. Privacy is a crucial factor for the acceptance of electronic payment systems [86]. Thus, a variety of P3 systems with different settings, privacy goals and adversary models has been proposed. A notable subclass of P3 systems are credit and Payment Channel Networks (PCNs) that do not only allow direct payments, but route the money over multiple intermediaries, similar to onion routing. We introduce the P3 systemsfurther in the following sections as needed. For an extensive informal overview, however, we refer the readers to [48].

High-Level Approach for P3 systems. Not only the protocol proposals, but also the majority of prior surveys that aim to compare private payment schemes come up with informal, ad-hoc security and privacy goals [2, 73, 48]. This complicates comparison, and differences between goals (“Strong privacy” and “Strong anonymity” [48]), assessments of fulfillment of the goals (“slightly high” vs “medium high” [2]) or even the meaning of goals (like “full unlinkability”) remain unclear. While these reports give a great first overview, we deem it necessary to have fine-grained, unified formal definitions to compare the approaches and solve the existing confusion around terms like “privacy”, “anonymity” and “pseudonymity” [80], as some of the informal works also conclude [2, 73].

To the best of our knowledge, the only formal approach to privacy in P3 systems is presented by Amarashinge, Boyen and McKague [4]. Their indistinguishability games are tightly tailored for cryptocurrencies, which allows them to model the adversary in a detailed fashion. For the privacy goal, they focus on exactly one entity (sender, receiver, value or meta-data) at the time and define an all-or-nothing protection for each. In a later work [3], the authors apply this formalization to analyze Bitcoin, Zcash, Monero and MimbleWimble in comparison to an idealized trusted third party solution.

Their highly specialized model is however not applicable to other P3 systems, like PCNs, and does not extend to other layers or combination of layers, as demonstrated to be highly relevant by recent attacks [134, 85]. Because the privacy goals, like hiding who payed whom, are however similar for all P3 systems and layers, we suggest to abstract from specific adversaries[3] and instead to be specific on the information that is hidden by the protocol, as well as on the restrictions of the anonymity set. Thereby, we represent the fine-grained, real world trade-offs in the protection much better than with the all-or-nothing approach of the state of the art.

7.2.2 Model

As in Chapter 3, we model *payment privacy notions* as games. The payment scenarios σ_0, σ_1 that the game attacker submits and then tries to distinguish now consist of payment tuples (s, r, v, aux) with a sender s, receiver r, value v and potentially auxiliary information aux, like e.g. a multi-hop route in a PCN. We use $\Diamond$ to express that no payment actually happens. Except for these changes, we use the game as introduced in Section 3.2. Depending on the use-case, the senders and receivers are, e.g., user identifiers or user pseudonyms, like wallets, and the length of a scenario can be restricted to correspond to one run of the protocol.

7.2.3 Notions

As before, we define our notions by the properties that are required to be hidden - and, in other words, which may differ in the scenarios that the adversary submits in the game.

Properties

Our properties showcase information desired to be protected or allowed to be leaked by the P3 system. For the selection of our properties we considered i) state-of-the-art P3 systems [63, 81, 110, 109, 114, 115, 137, 138] (see Appendix D.2.3) and ii) the systematic information leakage described in their ACN equivalent (see Section 3.3).

As for ACNs before (see Section 3.3), we define

- a property($\not\Diamond$) that ensures that the total number of events is equal,
- properties (E_S, E_R, E_V, E_{RV}) that allow only certain dimensions, like e.g. the senders, to differ between the scenarios,

[3]Note that the fine-grained adversary model is a different aspect of the protection than the privacy goal and that especially the model of [4] can be combined with our privacy goals for a detailed analysis of cryptocurrencies.

Table 7.3: Overview over our Payment Specific Properties

Property	**Meaning**
Σ_S	totals per sender are allowed to be learned
F_V	all values are fixed to the same value
F_{Σ_S}	all sending totals are fixed to the same value
$\|P\|_2$	total number of payments analyzed are 2
$\equiv_G$	graph (PCNs, credit networks) is allowed to be learned
$Atom$	considered payments happen simultaneously

- properties (U, Q, P, H) that fix the set of active senders, the sending frequencies, histograms and allow for user pseudonyms (similarly for receivers),
- properties (M_{SR}, M_{SV}, M_{RV}) that allow only the linking between two dimensions to differ in the scenarios, and
- a property (T_S) that only hides whether two payments have the same sender (similarly for receivers).

Specific for P3 systems. Values are sometimes paid as aggregates of transactions [81]. Some P3 systems thus might hide the values for single transactions, but allow the aggregated sum of the received or sent payment amounts per user to be learned, e.g. when this money is credited to or deposited from the user's account. We thus introduce the property Σ_S that allows the sender totals, i.e. the sum of payed values, to be learned (similarly for the receiver).

Definition 31 (Totals). *Let the function V_R calculate the total received amount for every receiver (V_S the sent amounts for every sender).*

$$V_R(\sigma_b, r') := \sum_{(s,r,v,a)\in\sigma_b \wedge r=r'} v, \qquad V_S(\sigma_b, s') := \sum_{(s,r,v,a)\in\sigma_b \wedge s=s'} v$$

Definition 32 ("Equal Totals" Σ_X). *For the scenarios σ_b:*

$$\Sigma_R : \quad V_R(\sigma_0, r) = V_R(\sigma_1, r) \qquad \Sigma_S : \quad V_S(\sigma_0, s) = V_S(\sigma_1, s)$$

Some P3 systems do not only allow the values of each payment to be learned (as expressed by E_{SR}), but even provide privacy only under payments with the same value[39, 81]. We model this with the property F_V that demands that the values of all payments, even *within a scenario*, are using the same fixed value, e.g. 1 coin. Further, for the totals per sender/receiver we introduce the same kind of restriction (F_{Σ_S}, F_{Σ_R}, $F_{\Sigma_S\Sigma_R}$), i.e. not only the total per user has to be the same in both scenarios, but indeed *all* users have to have the same (sending/receiving or sending and receiving) total.

Definition 33 ("Fixed X for all payments" F_X). *Let v be a value. For each payment $p_{b,k}$, resp. for the scenarios σ_b:*

$$\begin{aligned} F_V : &\quad p_{b,k} = (s_{b,k}, r_{b,k}, v, aux_{b,k}) \\ F_{\Sigma_S} : &\quad \forall s \in \mathcal{U} : V_{S(\sigma_0,s)} = V_{S(\sigma_1,s)} = v \\ F_{\Sigma_R} : &\quad \forall r \in \mathcal{U}' : V_{R(\sigma_0,r)} = V_{R(\sigma_1,r)} = v \\ F_{\Sigma_S\Sigma_R} : &\quad \forall s \in \mathcal{U}, r \in \mathcal{U}' : V_{S(\sigma_0,s)} = V_{S(\sigma_1,s)} = V_{R(\sigma_0,r)} = V_{R(\sigma_1,r)} = v \end{aligned}$$

As a simplification for the proof, some notions explicitly target a pair of two (most often simultaneous) payments [109], which we express with the property $|P|_2$. As the attacker gets to choose which two payments are used for the game, proving this simplified setting is sufficient to show that the uncertainty extends to *any pair* of such payments.

Definition 34 ("Exactly x payments" $|P|_x$). *Each scenario contains exactly x payments: $|\sigma_b| = x$*

Table 7.4: Payment Specific Notion Definition ($S\overline{O}$, $S\overline{O} - |U|$,$S\overline{O} - H$, $S\overline{O} - P$, $SF\overline{L}$, $SF\overline{L} - H$,$SF\overline{L} - P$, $(2S)\overline{O}$ are defined as for ACNs before.)

Name	**Properties**
Sender Unobservability Fixed Value ($S\overline{O} - F_V$)	$E_S \wedge F_V$
Sender Unobservability Leaking Graph ($S\overline{O}- \equiv_G$)	$E_S \wedge \equiv_G$
Sender Unlinkability ($S\overline{L}$)	$E_S \wedge Q$
Sender Unlinkability Fixed Value ($S\overline{L} - F_V$)	$E_S \wedge Q \wedge F_V$
Sender Unlinkability Fixed Total ($S\overline{L} - F_{\Sigma_S}$)	$E_S \wedge Q \wedge F_V \wedge F_{\Sigma_S}$
Sender Unlinkability Leaking Partition ($S\overline{L} - P$)	$E_S \wedge Q \wedge P$
(Sender-Value) Unlinkability ($(SV)\overline{L}$)	$\mathit{Atom} \wedge M_{SV}$
receiver notions	similarly to sender
Payment Unobservability ($P\overline{O}$)	-
Value Unobservability ($V\overline{O}$)	E_V
Value Unobservability leaking Graph ($V\overline{O}- \equiv_G$)	$E_V \wedge \mathit{Atom} \wedge \equiv_G$
Unobservability ($\overline{O}$)	E_{SRV}
Unobservability Fixed Value ($\overline{O} - F_V$)	$E_{SR} \wedge F_V$
Unlinkability Fixed Value ($\overline{L} - F_V$)	$E_{SR} \wedge F_V \wedge Q \wedge Q'$
Unlinkability Fixed Total ($\overline{L} - F_{\Sigma_{SR}}$)	$E_{SR} \wedge F_V \wedge Q \wedge Q' \wedge F_{\Sigma_S \Sigma_R}$
2-(Sender-Receiver) Unlinkability ($2(SR)\overline{L}$)	$\mathit{Atom} \wedge \|P\|_2 \wedge M_{SR}$

Specific to protocol choices. Finally, we introduce a set of properties for specific P3 system types that will all make use of the auxiliary information to express type specific choices for the payment privacy.

For PCNs and credit networks, like [110, 114], we define a variant of M_{SR}. In PCNs the auxiliary information describes the payment's path. We thus require that the paths of the mixed transactions have at least one honest node in common and that the alternative transactions of the second scenario switch the payment paths accordingly. Versions of this property limit the possible alternative paths to paths that are plausible for the chosen routing strategy, e.g. paths without loops.

Also, for PCNs or credit networks the resulting graph or parts of it might be disclosed [110, 114]. To ensure that the notion is not broken only due to different changes in the graph, we require with the property $\equiv_G$ that the final graph is equal[4] after both scenarios.

Definition 35 (Equivalence of Payment Scenarios). *We define equivalence ($\equiv$) of two payment scenarios as resulting in the same graph.*
Definition 36 ("Balanced" $\equiv_G$). *For the scenarios $\sigma_b : \sigma_0 \equiv \sigma_1$*

Further, sometimes privacy only stems from users doing an action at the same time [109]. In general the protocol model has to decide how the sequence of transactions in the scenario is translated into issuing times. However to discuss the specific case of concurrent payments, we define the atomicity property *Atom* that requires that all transactions of the scenario are happening simultaneously and thus the adversary can e.g. not win based on the order or timings of the transactions.

Definition 37 (Atomicity *Atom*). Π *processes all payments in σ_b atomically. No observations can be made in between payments.*

Payment Notion Hierarchy

We define notions as combinations of properties according to Table 7.4. Our selection is tailored to include the goals of state-of-the-art P3 systems [63, 81, 88, 110, 109, 114, 115, 137, 138] (as discussed in Appendix D.2.3) and goals that were shown to be important in the related area of anonymous communication, while at the same time conforming to intuitive, basic goals.

[4]Versions of this property can require only partial knowledge about the graph to be equal.

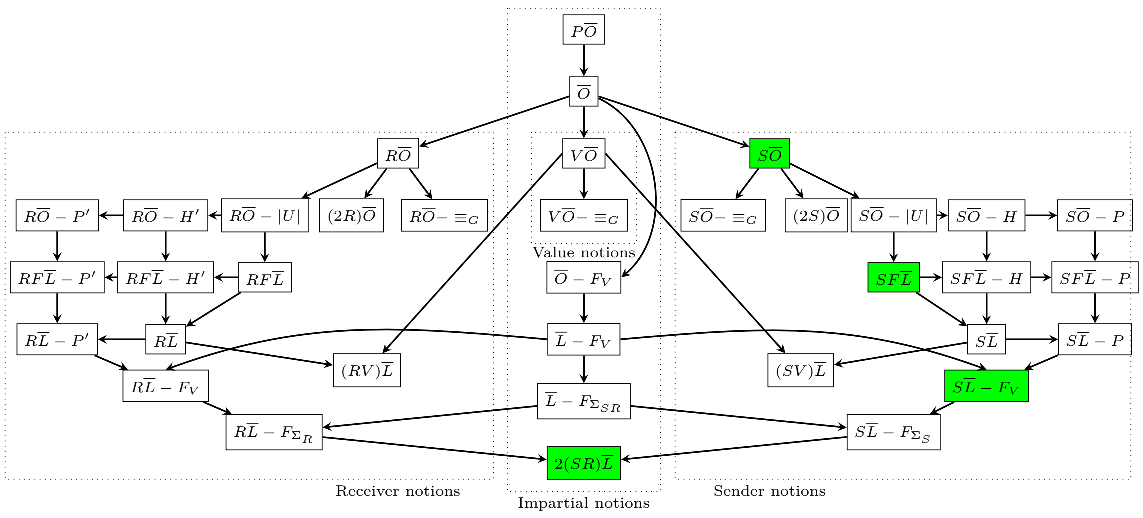


Figure 7.2: Hierarchy of privacy notions. The notion at the start of an arrow is strictly stronger than the notion at the end of that arrow. This "stronger"-relation is transitive. The green, highlighted notions are later used for our anonymity set size simulation.

As in Chapter 3 for anonymous communication notions, some payment notions are strictly stronger than others. We provide the proof sketches for such relations in in Appendix D.2.1 and depict the final implications between our payment notions as a hierarchy in Figure 7.2.

Notion Overview. We group the notions into sender notions that focus on protecting senders, receiver notions that focus on protecting receivers, and impartial notions that protect senders and receivers equally well.

Within the *impartial notions*, we define a subgroup that specifically focuses on protecting the paid values. The strongest of these notions, *Value Unobservability*, hides the values completely, i.e. the adversary can no longer observe that a certain value was payed (by anyone). The weaker value notion, *Value Unobservability leaking Graph*, is specific to payment channel/credit networks and requires that the resulting (credit) graph can be learned by the adversary. As part of the impartial notions the overall strongest notion *Payment Unobservability* prohibits any information to be leaked, whereas for *Unobservability* the adversary can learn how many transactions happen in total (and about auxiliary information, like fees, if any exist), but nothing else. The next weaker notion (*Unobservability Fixed Value*) specifies protection of all information about the sender and receiver, but allows to learn that all transactions have the same fixed value. The two weaker Unlinkability notions allow the sending and receiving frequencies as well as the fixed values (*Unlinkability Fixed Value*) or the fixed values and totals of all incoming and outgoing payments per user (*Unlinkability Fixed Total*) to be learned. The weakest impartial notion (*2-(Sender-Receiver) Unlinkability*) only specifies that for two transactions the adversary cannot tell which of the involved senders payed which involved receiver.

The group of *sender notions* concentrates on the senders (or observations about them). For the strongest of these notions (*Sender Unobservability*) anything about values and receivers can be learned, but nothing about the senders. For weaker notions, the sender protection can however be restricted by the activities, frequencies, histograms, graph knowledge and fixed values for all as introduced before (*area spanning* $S\overline{O} - |U|$ to $S\overline{L} - P$ and $S\overline{O} - \equiv_G$). *Sender Unlinkability Fixed Value* additionally limits values and *Sender Unlinkability Fixed Total* totals to be equal. $(2S)\overline{O}$ further expresses whether the adversary can determine if two payments belong to the same sender, without necessarily learning who this sender is, and *(Sender-Value) Unlinkability* hides only the relationship between sender and paid values.

The hierarchy of receiver notions resemble the sender notions correspondingly.

7.2.4 Anonymity Sets for Realistic User Behavior

In this section we perform an initial investigation of the anonymity set sizes that are actually to be expected in practice. As they rely on specific payment choices, we model realistic user behavior and simulate actual payments accordingly for our analysis.

We study the anonymity sets corresponding to the sender notions Sender Unobservability, Sender Frequency Unlinkability and Sender Unlinkability Fixed Value empirically, to then investigate their sizes under realistic, generated payment traffic. For this selection, we chose notions that i) are used by P3 systems as their goal (see Appendix D.2.3 for details) and ii) are covering a large part of the sender notions in terms of their strength. It indeed is hard to decide on realistic parameters, especially if sender-receiver pairs, or a specific payment network topology are needed - as for instance in path-based anonymity sets. We hence first study anonymity sets that we can determine without such information, and later study an example including path-based anonymity sets on real data. Considering that some of the anonymity sets are expected to be prohibitively small in several cases, we extend our study to explore preliminary strategies that may help increase the anonymity set sizes at limited cost.

Setup and Preparation

Anonymity Sets. First, we transform the chosen goals to real world anonymity sets. Sender Unobservability requires that two scenarios that differ only, but arbitrarily, in the senders, are indistinguishable. Thus, any user that participates in the protocol cannot be distinguished from the real sender and the anonymity set includes *all users*.

Sender Frequency Unlinkability requires additionally that the compared scenarios have the same set of active senders. Thus, any user that sent something during the observed payments cannot be distinguished from the real sender and the anonymity set includes *all active senders*.

Sender Unlinkability Fixed Value requires further that any payment of the scenario has the same value and that the senders are merely permuted between the scenarios (i.e. partitioning the payments per sender has to result in the same partitions in both scenarios). Interestingly this does not allow the same freedom in placing senders in the alternative (second) scenario as the two notions discussed earlier: We cannot only replace the sender of a single payment, but have to do the same replacement potentially for multiple payments that are linked to the same sender.[5] There are still indistinguishable scenarios that replace the sender of the multiple payments with any other active sender. The sender anonymity set hence differs just due to the fixed value; it thus includes *all active senders that send the same value.*

Given the diversity of possible payment values we expect the anonymity set sizes for the corresponding notions to be very small. We are hence interested, to which extent the senders may be able to increase their privacy by adapting their behavior and investigate potential strategies in our evaluation. Precisely, we use the intuitive approaches i) to increase the transaction value to the next higher value that is known to be used by others, ii) to delay transactions to increase the chance for some additional payments with identical value to be created and iii) to generally restrict all transaction values to specific, allowed value classes.

Setting. To identify the impact of deciding on a different notion, we calculate anonymity set sizes for each notion with a round based python simulation. All payments that are initiated within an epoch time slot are considered to be mixed with each other for the *active* or *value anonymity set*. This models e.g. a tumbler with fixed times to output payments again. We expressly have not implement the tumbler (or any other P3 system), but merely draw payment events according to the user model and count the number of distinct users with the considered criteria in each

[5]Notice that this difference is not reflected in the sender anonymity set of a single payment, but instead represents another leakage, namely: *which payments belong to the same sender*, independent of *who the sender is.*

epoch, assuming that they will choose the same tumbler (or corresponding anonymization component).

Data. We report on the results based on a group size of 100 000 users for our experiments, performing 30 repetition for each parameter set. Repeating the experiments with 10 000 and 1 000 000 users confirmed our findings.

We modeled our assumptions based on [117], which measured JoinMarket, a trading platform for privacy enhanced payments with Bitcoin. The values consequently are drawn following a lognormal distribution with mean of 84 USD and standard deviation of 2.4. Thus, the quantiles are 16 USD (25%), 84 USD (50%) and 420 USD (75%). We rounded all values to full USD to improve the chances for the value to match an existing anonymity set, and the resulting transactions cover values from one to several million USD as in the published data.

Information about senders is sparse as the networks try to protect the senders' anonymity. We thus use the best guess we can make for sending behavior: inter-sending times that are distributed according to a Poisson distribution. To account for the uncertainty, we vary the parameter λ $(10, 50, 100)$ and to give them a real world meaning, we match $\lambda = 50$ to real world time according to the total payments made per day in different networks as shown in Table 7.5. So $\lambda = 50$ represents the current payment situation, $\lambda = 10$ the situation that the payment networks are used much more frequently and $\lambda = 100$ the situation that the networks are used less frequent. We ensure that the simulations reach a stable state before we measure the anonymity sets.

Table 7.5: Real world time per simulation time unit (assuming different usage frequencies)

Usage frequency	**Time per simulation time unit**		**Payments per day**
	Generated Data	**Ripple Data**	
Ethereum [67]	2.5 min	3.8 sec	1.2 mio
Bitcoin [120]	9.5 min	15 sec	0.3 mio
Germany (incl. cash)	1.7 sec	45 millisec	99.6 mio
Canada (incl. cash)	7.5 sec	200 millisec	22.7 mio

Results.

We first compare the anonymity sets that result directly from our data, before we then adapt the paying strategies to increase the value-based anonymity sets. Our intuitive adaptions cover the individual increase of the paid value and waiting time, as well as a cooperative strategy to build common value classes.

Anonymity Set Without Modified Behavior. Table 7.6 shows an overview of the distributions of anonymity set sizes. Increasing either epoch length or payment frequency results in growing anonymity sets, as we would expect.

Table 7.6: Lower quartile, median and upper quartile for anonymity set sizes under varying frequencies and epochs (active sizes rounded)

Frequency \ **Epoch**	**12.5 min**	**25 min**	**125 min**
high: active	45.2k **45.3k** 45.4k	83.6k **83.7k** 83.8k	100k **100k** 100k
active+value	1 **1** 3	1 **1** 4	1 **1** 4
normal: active	10k **10k** 10.1k	20.2k **20.3k** 20.3k	93.5k **93.6k** 93.6k
active+value	1 **1** 3	1 **1** 3	1 **1** 4
low: active	6.2k **6.2k** 6.3k	12.6k **12.6k** 12.7k	54.9k **55k** 55k
active+value	1 **1** 2	1 **1** 3	1 **1** 3

Comparing the effective size of anonymity sets with matching values to the overall number of users (100 000) yields sobering results. The same holds for the number of simultaneously active users

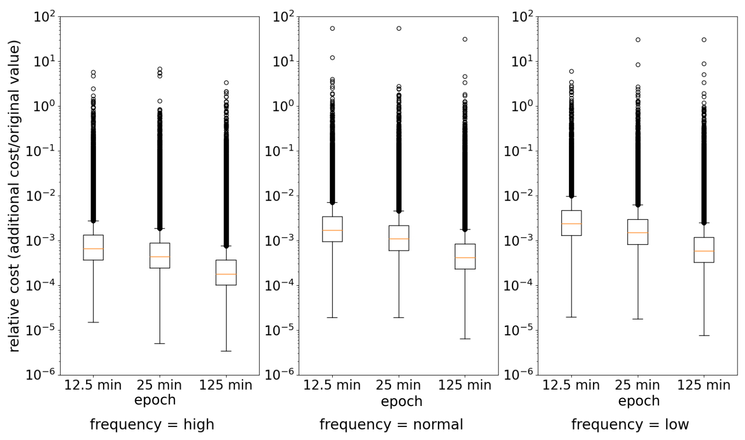


Figure 7.3: Relative cost to not be unique under epoch times of 12.5, 25 and 125 minutes and varying frequencies, i.e. $\frac{\text{additional cost}}{\text{desired value}}$

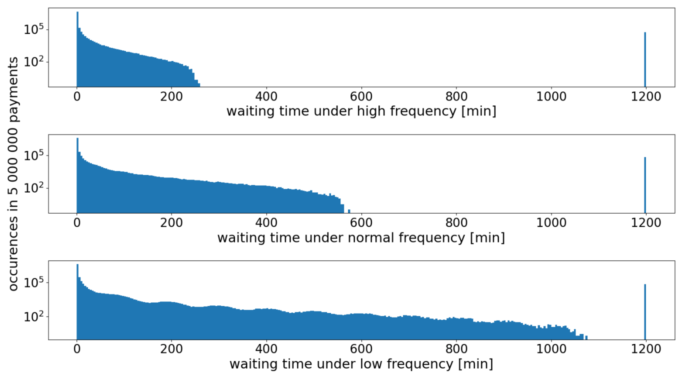


Figure 7.4: Waiting time until another payment with the same value occurs investigated for 5 million generated payments. 1200 represents that no such payment occurred in the generated payment setting. For the high frequency waiting times up to 250 minutes and under low frequency up to 1100 minutes occur.

given expected payment frequencies and epoch times at or below 25 minutes. Both decrease when epoch times or payment frequencies are reduced, as expected.

The anonymity set sizes for the value-based sets clearly are unacceptably small, with about 3 000 entirely deanonymized payments on average. The number of active senders remains on the order of tens of thousands of users, which may be considered acceptable as potential anonymity set size.

Paying more against deanonymization. If a user would know all payment values of an epoch and decide to increase their payment to reach the next higher value payed during the same epoch, she could create a value based anonymity set of at least 2. The distribution of relative cost over all additional costs > 0 is shown in Figure 7.3. The additional cost to ensure that at least two users share the same value in most cases ranges around a few to a few dozen percent of the original value. Since very high payment values do occur with small probability, there exist a few very costly outliers.

Waiting longer against deanonymization. With perfect knowledge of all payments, another strategy is to delay a transaction in order to find an anonymity set of at least size 2. We report the necessary waiting time per user in Figure 7.4. The time to wait depends highly on the usage frequency. But even with the highest usage frequency it remains over 2 hours for a substantial

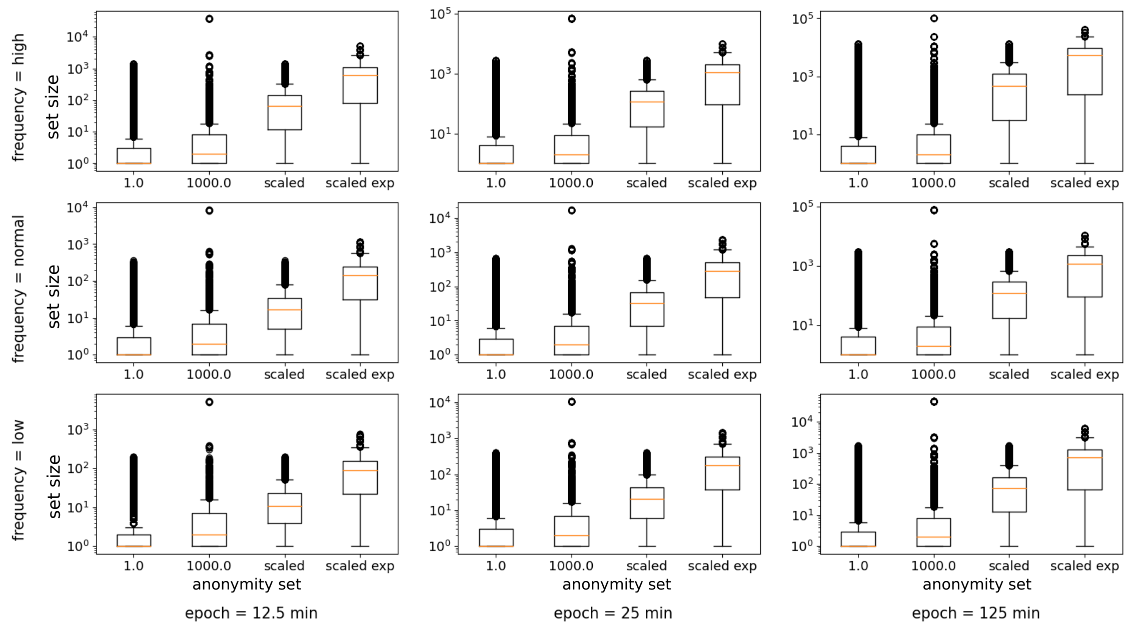


Figure 7.5: Anonymity set sizes with value buckets. The results demonstrate a great improvement for all epoch times and frequencies when using the scaled buckets that at the same time also cause lower relative cost than fixed bucket sizes.

part of the payments.

Introducing Value Buckets. Consider all users to participate in a cooperative strategy: Reducing the number of possible payment values to choose from increases the chance of collisions, and hence the expected anonymity set size.

A simple approach is to use fixed value buckets, for instance to round the values up to full 10-, 100-, or even 1 000-dollar amounts. In the last case of course a user wanting to buy something for 1 dollar would need to pay 1 000. Another option is to reduce the relative additional incurred cost, i.e. the amount of money a user has to pay divided by the intended value. We hence investigate two types of scaling buckets: 1) a cheap variant that limits the relative additional cost to 10% of the original value and 2) a more expensive variant that limits the relative additional cost to 100% of the original value, but still achieves a much better relative cost than the above strategy of rounding to full USD 1 000 amounts.

Our cheap scaling buckets assume the new value (v_{cheap}) to be calculated from the original value (v_{ori}) as follows:

$$v_{cheap} = \left\lceil \frac{v_{ori}}{10^{\lfloor \log_{10}(v_{ori}) \rfloor - 1}} \right\rceil \cdot 10^{\lfloor \log_{10}(v_{ori}) \rfloor - 1}$$

This means that all values up to USD 100 are rounded to full 1-dollar values, from 100 and 1 000 to full 10-dollar values, from 1 000 to 10 000 to full 100-dollar values and so on. The expensive strategy reduces the number of buckets further, by scaling them to a magnitude higher values (v_{exp}), e.g. to full 10-dollar values between 1 and 100, etc., as follows:

$$v_{exp} = \left\lceil \frac{v_{ori}}{10^{\lfloor \log_{10}(v_{ori}) \rfloor}} \right\rceil \cdot 10^{\lfloor \log_{10}(v_{ori}) \rfloor}$$

The anonymity sets per epoch are shown in Figure 7.5. While some deanonymized payments remain (on average 27.4 (cheap), 1.96 (expensive)), the scaling buckets perform well with anonymity set sizes of mostly at least 10 users - a huge improvement from the original values (nearly 3 000 deanonymized payments) and especially much better than fixed bucket sizes (still 63.7 deanonymized payments for 1 000-dollar buckets). Thus, with scaling buckets there is hope for the value-based

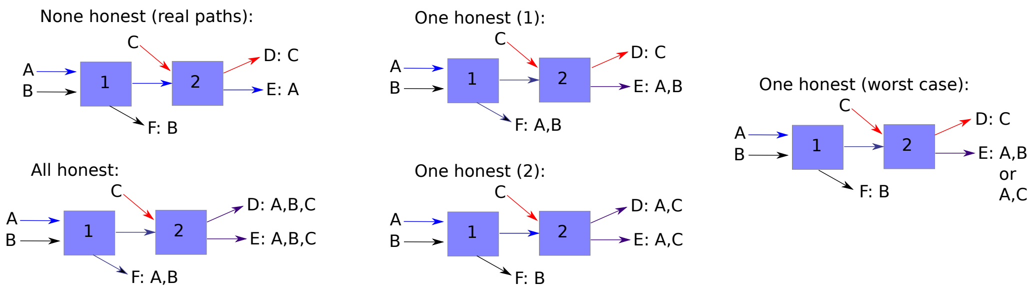


Figure 7.6: Maximal (all nodes honest) and minimal anonymity sets (worst case node per path is honest) illustrated.

anonymity sets, even though they of course are still rather small and should be used with caution.

7.2.5 Simulation for path-based multihop protocols

We then turn to path-based multihop protocols, and investigate the expected anonymity set size given our privacy notion 2-(Sender-Receiver) Unlinkability. General information about sender-receiver relationships and payment paths is too sparse to synthesize this data. We hence used real world Ripple data for this study and validate our findings for the other anonymity sets with real world data at the same time.

Setup and Preparation

Anonymity Sets. We use the anonymity sets from Section 7.2.4 and additionally introduce path-based anonymity sets for 2-(Sender-Receiver) Unlinkability. 2-(Sender-Receiver) Unlinkability requires that only two payments differ between the scenarios by exchanging the sender and receiver pairs. For PCNs we even require that the payments meet at an honest node on the path. Thus, any sender with whom a transaction can be exchanged this way is indistinguishable to the real sender for an adversary. For our simulation, we are choosing a routing algorithm that does not include any loops and hence by default use the modified version that does only consider alternatives without a loop in the routing path. (We show the effect of including loops in Appendix D.2.2.)

Further, the anonymity sets for 2-(Sender-Receiver) Unlinkability depend on the adversary model as the notion considers only paths with a common *honest* node. We will be looking at the two (non-trivial) extreme cases below (as illustrated in Figure 7.6): (i) all nodes are honest and (ii) exactly one node per payment path is honest. Thus the first gives us the maximal anonymity set reachable for this setting. For the second case, we decide on the worst intermediate node, i.e. the one with the least other payments, to be honest. Thus this is the minimal anonymity set (except for the unprotected case when there is no honest node on the path, and the anonymity set trivially only includes the real sender).

Setting. In addition to the epoch time of Section 7.2.4, we introduce another type of time slot: the *hop time* as the delay per single hop. All payments that are initiated within a hop time slot and are at the same intermediate node are considered to be mixed for the *value and path anonymity set*. As there might be multiple intermediate hops on a path and we like the worst case time to complete a payment to be similar to the one in the other setting, we decide on shorter hop times than epoch times, but also vary them until both times are equal.

Data. We use Ripple payment data from January 2013 until August 2016, provided by [136]. *Ripple* [7] is a credit network [69, 51]. In a credit network participants allow other participants a credit based on the trust between the entities. This results in a graph where vertices represent participants and edge weights the allowed credit. Similar to Bitcoin, a publicly available global ledger records the transactions and is updated upon consensus.

The dataset contains over 800 000 payments with sender, receiver, value and time stamp. Those payments already have self-transactions filtered out. Further the data contains the payment channel graph and updates for the graph over the same time interval, with graph updates missing for some time spans. As there are on average only 25 payments per hour, we decide to increase time by a factor of 1 000, i.e. each second is interpreted as a millisecond, to work with payment data that is more realistic for current user behavior. Note that the number of payments per time varies greatly from 6.19 to around 70 payments/minute later. We thus use the payments of the last 10 months (October 2015 - August 2016) to avoid effects due to the novelty of the network in earlier years, resulting in 52 payments/minute on average (see Table 7.5 for a comparison to current payment behavior).

To generate a path for each payment, we calculate the shortest path between sender and receiver that has enough capacity to allow for the current payment and update the capacities accordingly, before looking at the next payment. This strategy results in nearly 238 000 successful payments of which nearly 182 000 use at least one intermediate hop and the average number of intermediate hops is 1.23 with a maximum of 10 intermediate hops.

Results

We quickly discuss the outcome of the non-path based cases for the Ripple data set, before we investigate the influence of path-based restrictions.

Anonymity Sets Without Path Considerations. Evaluating the strategies from above on the Ripple data without paths, we observe that the anonymity sets for active senders and scaled value buckets for an epoch of 25 minutes are much smaller than for the generated data above (cmp. the first three box plots of Figure 7.7).

Path-based Effects. With regards to the path-based approaches we want to ensure a total processing time that roughly equals 25 minutes or less for each payment, as above. Given the maximum path length of 10 hops in our experiment, we decided to set the hop time to 2 minutes such that also payments with the maximum path length are completed on time. Despite this strict choice, the path-based anonymity sets (ignoring the payment values) are surprisingly high, even though there are only 1.23 intermediate nodes per payment path on average (cmp. box plots 4 and 5 in Figure 7.7). We gather that many payments are using few, popular intermediate nodes, traversed by many simultaneous transactions. Indeed, depending on the hop time, as little as 12-18 nodes are needed to be honest in total, to have at least one honest node on every path in the given data set.

Testing for the worst case and choosing the least suitable node on each path to be honest, this number increased to 150 different honest nodes. Note that if there would always be exactly one intermediate node on the path the min and max path-based anonymity sets would be equal. That there is often only one intermediate hop is likely the reason for the good performance even of the min path-based anonymity sets.

Note that, theoretically, the maximum path-based anonymity could yield higher anonymity sets than counting simultaneous active senders: The epochs (used for the active senders) use a fixed time window (e.g. 25 minutes), but the time window in which payments contribute to the maximum path based anonymity sets might exceed the time a single payment needs (e.g. 25 minutes). The reason is that a payment at the very end of its processing (say e.g. at its minute 20) may still mix with a payment that has just begun processing. In this way both; payments starting up to 25 minutes before the payment in question and those starting 25 minutes after it, potentially contribute to the payment's anonymity set. This effect is even amplified by the transitivity of the

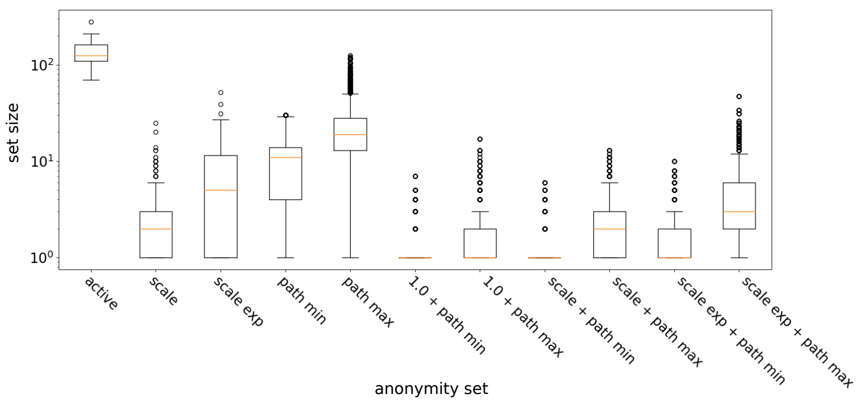


Figure 7.7: Hoptime of 2 minutes. The path-based sets are performing surprisingly well due to the existence of few, very popular nodes at which the transactions get mixed. Leaking values however limits the anonymity sets drastically.

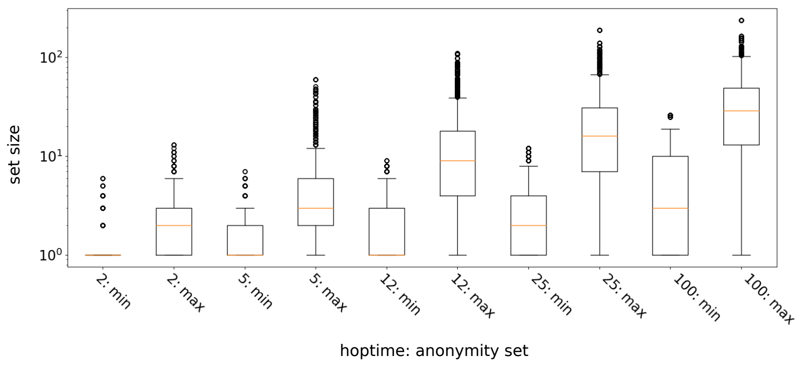


Figure 7.8: Path anonymity set sizes for the scaled buckets with increasing hop times. As expected with higher hoptime, i.e. latency, the anonymity set sizes grow, however for the minimum anonymity set sizes the numbers are still very small.

maximum path based anonymity set. However as the average path length is short in our data, we do not observe such effects in Figure 7.7.

Let us consider both the path and value, as required for the weakest notion under investigation, next. This naturally reduces the anonymity set size drastically, as seen above. Indeed, taking the worst case node choice for each path, we observe that not even the expensive scaling bucket strategy can effectively save the situation (cmp. Figure 7.7, 'scale exp + path min'). Only a very small minority of transactions is effectively anonymized in this worst case scenario.

Finally, in Figure 7.8 we further investigate the effect of varying the hop times. The anonymity sets grow as expected if higher latency is accepted. However even with tremendous latency (up to 16.67 hours for a 10 hop path) the average size of the minimum path based anonymity sets remains as low as 3.

Summary. We observed that the expected anonymity set sizes given realistic assumptions are unacceptably low for many of the systems. We hence investigated suitable strategies to improve the situation. Our results show that delaying transactions may yield low benefit. Increasing payment values to rounded classes on the other hand does not impose prohibitive additional cost with scaled buckets, and helps to get closer to acceptable anonymity set sizes.

8. Conclusion

With the help of existing indistinguishability games and differential privacy, we defined an extensive set of over 50 privacy goals for anonymous communication and compared them to create our hierarchy of privacy notions. Indeed, that many goals are necessary as the protection that is required in different situations is extremely diverse (see Appendix A.6 for examples). Our definitions counter the ambiguity and incomparability of many practically used privacy goals and allow for a precise analysis of ACNs.

With this formal foundation, we were able to produce the following results:

1. We resolved naming conflicts around the informal protection goal of "sender anonymity" (Section 3.8), which was mapped to our Sender Unobservability $S\overline{O}$ and also to our (considerably weaker) Sender-Message Unlinkability $SM\overline{L}$ in different related works.

2. We stated more precise privacy goals for and compared the assumed protection of different existing performance bounds (Appendix B.1, especially Figure B.1). Precisely, the stated performance overhead of the Trilemma is indeed not only necessary for the strong Sender Unobservability $S\overline{O}$ goal, but already for Sender-Message Pair Unlinkability $(SM)\overline{L}$, one of the weakest notions in our hierarchy. Similarly, the overhead stated in the Dropping-Bound is not only necessary for Communication Unobservability $C\overline{O}$, the strongest notion of our hierarchy, but already for Sender-Receiver Pair Unlinkability $(SR)\overline{L}$, one of the weakest notions.

3. We investigated ACNs regarding our notions to identify basic techniques and thereby learned about their inner workings (Section 3.7 and Appendix A.9). During our investigations, we found several vulnerabilities and attacks on existing networks:

    - Deriving the path length in Sphinx[1] (Section 5.2.1); Because of the identifiable instead of random header padding, the exit relay can derive the length of the routing path from the length of the padding.

    - The malleability attack on Sphinx'[2], the fixed Minx', HORNET's and Loopix' Sender-Receiver Pair Unlinkability $(SR)\overline{L}$ (Section 5.3.1); Because of the missing hop-by-hop integrity protection of the payload, the adversary can modify the payload of the packet when it is sent from her victim and recognize the delivery of the unusual message at the adversarial receiver.

    - The malleability attack on HORNET's Sender-Message Pair Unlinkability $(SM)\overline{L}$ (Section 5.3.1); Because the chosen encryption scheme does not bi-directionally distribute errors, the above attacker can modify only a part of the payload – enough to recognize the message as unusual – and learns the other part of the sender's original message.

    - Frequency attacks ($SF\overline{L}, RF\overline{L} - P'$) on Loopix (Appendix A.9.4); Because communication frequencies are not padded to a maximum, the adversary can infer imbalances between the frequencies in some extreme cases.

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

[2] $(SR)\overline{L}$ breaks in an unintended setting and in the intended setting senders and exits nodes can be linked.

- Attacks derived from the performance bounds (Section 4.3.1) allow further conclusions about non-achieved notions for Tor, HORNET, Threshold-Mixes, Herd, Dicemix, Dissent, Loopix, Riposte, Riffle and Vuvuzela. We stress however that (nearly all of) these networks did not aim to achieve the attacked notions for the considered adversary models.

Motivated by the malleability attack, we found and corrected flaws in the underlying proof strategy of Camenisch and Lysyanskaya in Chapter 5, as well as extended it for replies in Chapter 6. Thereby we provide reusable, game based security properties for future work. With the help of this foundation, we were further able to produce the following results:

1. We proved the adapted version of Sphinx from Beato et al. to be secure by showing our game-based properties (Section 5.4). This version does however not support replies.

2. We provide two secure packet formats for *repliable* onion routing (Section 6.5 and 6.7). Thereby, we solve the challenge of implicit payload protection, which arises from the combination of reply support and the malleability attack, with the help of SNARGs and updatable encryption.

Beyond anonymous communication in Chapter 7, we supported proximity tracing applications, as well as payment networks with a formal hierarchy of privacy goals and an initial investigation of proposed solutions. For payments our study of real world data further gives first insights into the expected anonymity set sizes, as well as proposes strategies to increase the value-based anonymity sets.

In summary, we provide useful theoretical foundations for and beyond anonymous communication and derive first practical contributions from them. The extended application of our formal groundwork, both by analyzing the privacy of more protocols and by using our onion routing properties for security proofs, is the most obvious direction of future research. Besides the mentioned tangible outcomes, this thesis further represents an important step towards the complete understanding of anonymous communication, which is an interesting, yet ambitious long-term objective for future work.

More precisely, recall that the situation of anonymous communication is extremely complex. Every solution offers a trade-off between multiple, non-linear dimensions: the privacy protection, adversary, performance requirements, other security goals and environmental assumptions.

While we provide an extensive, in-depth investigation of privacy goals in Chapter 3, we still continue to find extension possibilities even in this dimension (see [100]). Moreover, the other dimensions are to the best of our knowledge not even understood on a comparable level. The adversary model combines multiple dimensions in itself. An adversary might corrupt different parts of the network, actively or passively for a short or longer time period and have further restrictions. A large number of different adversarial models has thus potentially to be considered for each privacy goal. Performance requirements in different use cases for communication are similarly diverse. For example video calling requires a much lower latency than emailing and mobile phones offer only especially limited resources. Additionally, other security goals, like availability or (a limited form of) accountability, might be advisable for some situations. Environmental assumptions, like the (non-)existence of a public key infrastructure or (secure) auxiliary protocols further influence the applicable solutions.

Thus, even the problem space of anonymous communication is hard to describe completely with the existing knowledge and formalization. Further, we are of course even more interested in the part of the problem space for which it is actually possible to find solutions. Our investigation of existing performance bounds in Chapter 4 provides an overview of the limited knowledge that we currently have regarding the (im-)possibilities in this problem space. As the space is however that large and complex, future work has much more to discover.

Naturally we are not only interested in discovering the parts for which solutions (possibly) exist, but in the actual solution, i.e. the used techniques and the constructed networks for any concrete point in the problem space. With our first consideration of basic building blocks against the global passive adversary in Section 3.7, we provide initial insights for a small subspace of the complex problem space.

The quest to understand this fascinating multifaceted problem and its solutions, is of course useful to build better networks for a specific use case with the improved knowledge. Furthermore it opens the door for more far-reaching changes in the anonymous communication research. If we indeed sufficiently understand the problem space, corresponding solution techniques and how they compose, we can exploit this knowledge to build assistance for the privacy (and security) proofs, or even automate them. Thereby a new level of certainty and efficiency in designing secure and private solutions can be reached.

Even further, the understanding of techniques and their composition might not only allow us to find one static solution for each use case, but to build adaptive solutions. By switching some of the employed protection techniques according to the users' needs in their current environment, we can optimally protect them while they communicate in our fast-paced world.

Taking a holistic view, not only on anonymous communication, but on all privacy enhancing technologies and understanding their relations to each other is another major direction for future research. If we can understand the relations between different areas of privacy research in detail, we are able to transfer results between them. With our work on similar privacy definitions for proximity tracing and payment systems in Chapter 7, as well as with our comparison of ACN notions to existing physical layer security notions [106], we make the first steps in this direction. Extending these connections, opens the door for a new understanding of the protection in each of these areas and for privacy in general.

Bibliography

[1] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, 2001.

[2] N. Amarasinghe, X. Boyen, and M. McKague. A survey of anonymity of cryptocurrencies. In *Australasian Computer Science Week Multiconference*, 2019.

[3] N. Amarasinghe, X. Boyen, and M. McKague. The complex shape of anonymity in cryptocurrencies: Case studies from a systematic approach. *FC*, 2021.

[4] N. Amarasinghe, X. Boyen, and M. McKague. The cryptographic complexity of anonymous coins: A systematic exploration. Technical report, ePrint, https://eprint.iacr.org/2021/036.pdf, 2021.

[5] M. Ando and A. Lysyanskaya. Cryptographic shallots: A formal treatment of repliable onion encryption. *eprint, https://eprint.iacr.org/2020/215.pdf*, 2020.

[6] M. Ando, A. Lysyanskaya, and E. Upfal. On the complexity of anonymous communication through public networks. *arXiv preprint, arXiv:1902.06306*, 2019.

[7] F. Armknecht et al. Ripple: Overview and outlook. *Trust and Trustworthy Computing*, 2015.

[8] E. D. Ayele. Analysis and deployment of the BitTorrent protocol for Community Ad-hoc Networks. Technical report, TU Delft, 2011.

[9] M. Backes, I. Goldberg, A. Kate, and E. Mohammadi. Provably Secure and Practical Onion Routing. In *Computer Security Foundations*, 2012.

[10] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi. Anoa: A framework for analyzing anonymous communication protocols. *Computer Security Foundations*, 2013.

[11] 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.

[12] M. Backes, P. Manoharan, and E. Mohammadi. Tuc: Time-sensitive and modular analysis of anonymous communication. In *Computer Security Foundations*, 2014.

[13] E. Balkovich, D. Prosnitz, A. Boustead, and S. C. Isley. *Electronic Surveillance of Mobile Devices.* Rand Corporation, 2015.

[14] F. Beato, K. Halunen, and B. Mennink. Improving the sphinx mix network. *Cryptology and Network Security*, 2016.

[15] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. *Asiacrypt*, 2001.

[16] M. Bellare, A. Boldyreva, L. Knudsen, and C. Namprempre. Online ciphers and the hash-cbc construction. *Crypto*, 2001.

[17] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. *Crypto*, 1998.

[18] A. Berke, M. Bakker, P. Vepakomma, R. Raskar, K. Larson, and A. S. Pentland. Assessing Disease Exposure Risk with Location Data: A Proposal for Cryptographic Preservation of Privacy. Technical Report arXiv:2003.14412 [cs], 2020.

[19] 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.

[20] B. Bernabe et al. Privacy-preserving solutions for blockchain: Review and challenges. *IEEE Access*, 2019.

[21] O. Berthold, H. Federrath, and S. Köpsell. Web mixes: A system for anonymous and unobservable internet access. In *Designing privacy enhancing technologies*, 2001.

[22] W. Beskorovajnov, F. Dörre, G. Hartung, A. Koch, J. Müller-Quade, and T. Strufe. Contra corona: Contact tracing against the coronavirus by bridging the centralized-decentralized divide for stronger privacy. *ePrint, https://eprint.iacr.org/2020/505.pdf*, 2020.

[23] M. Bhargava and C. Palamidessi. Probabilistic anonymity. In *Concurrency Theory*, 2005.

[24] N. Bitansky et al. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. *ITCS*, 2012.

[25] M. R. Bloch and J. Barros. *Physical Layer Security From Information Theory to Security Engineering.* Cambridge University Press, 2011.

[26] J.-M. Bohli and A. Pashalidis. Relations among privacy notions. *TISSEC*, 2011.

[27] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan. Key homomorphic prfs and their applications. *Crypto*, 2013.

[28] D. Boneh, E. Shen, and B. Waters. Strongly unforgeable signatures based on computational diffie-hellman. *PKC*, 2006.

[29] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh. Zether: Towards privacy in a smart contract world. In *FC*, 2020.

[30] C. Cadwalladr and E. Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The guardian*, 2018.

[31] J. Camenisch and A. Lysyanskaya. A formal treatment of onion routing. *Crypto*, 2005.

[32] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. *FOCS*, 2001.

[33] R. Canetti, H. Krawczyk, and J. B. Nielsen. Relaxing chosen-ciphertext security. *CRYPTO*, 2003.

[34] R. Canetti, A. Trachtenberg, and M. Varia. Anonymous collocation discovery:harnessing privacy to tame the coronavirus. Technical Report arXiv:2003.13670, 2020.

[35] C. Castelluccia, N. Bielova, A. Boutet, M. Cunche, C. Lauradoux, D. L. Métayer, and V. Roca. Desire: A third way for a european exposure notification system. Technical report, 2020. https://github.com/3rd-ways-for-EU-exposure-notification/project-DESIRE/blob/master/DESIRE-specification-EN-v1_0.pdf [Online; last accessed April-2020].

[36] D. Catalano, M. Di Raimondo, D. Fiore, R. Gennaro, and O. Puglisi. Fully non-interactive onion routing with forward secrecy. *International journal of information security*, 2013.

[37] D. Catalano, D. Fiore, and R. Gennaro. A certificateless approach to onion routing. *Journal of Information Security*, 2017.

[38] Chainalysis: The blockchain analysis company. `https://www.chainalysis.com`, 2014.

[39] D. Chaum. Blind signatures for untraceable payments. *Advances in cryptology*, 1983.

[40] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1988.

[41] D. Chaum, F. Javani, A. Kate, A. Krasnova, J. de Ruiter, A. T. Sherman, and D. Das. cMix: Anonymization by high-performance scalable mixing. *USENIX Security*, 2016.

[42] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.

[43] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. HORNET: High-speed onion routing at the network layer. *ACM CCS*, 2015.

[44] 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.

[45] H. Chen, Y. Xiao, X. Hong, F. Hu, and J. Xie. A survey of anonymity in wireless communication systems. *Security and Communication Networks*, 2009.

[46] H. Cho, D. Ippolito, and Y. W. Yu. Contact Tracing Mobile Apps for COVID-19: Privacy Considerations and Related Trade-offs. Technical Report arXiv:2003.11511, 2020.

[47] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *IEEE FOCS*, 1995.

[48] M. Conti et al. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys & Tutorials*, 2018.

[49] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. *IEEE S&P*, 2015.

[50] P. Cuff and L. Yu. Differential privacy as a mutual information constraint. *ACM SIGSAC*, 2016.

[51] P. Dandekar et al. Strategic formation of credit networks. *WWW*, 2012.

[52] G. Danezis and I. Goldberg. Sphinx: A compact and provably secure mix format. *IEEE S&P*, 2009.

[53] G. Danezis and B. Laurie. Minx: A simple and efficient anonymous packet format. *WPES*, 2004.

[54] D. Das et al. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two. *IEEE S&P*, 2018.

[55] D. Das et al. Not all is lost for anonymity - but quite a lot is. coordination among users can help anonymity. *HotPETs*, 2019.

[56] Y.-A. de Montjoye et al. Unique in the Crowd: The privacy bounds of human mobility. *Sci. Rep*, 2013.

[57] J. P. Degabriele and M. Stam. Untagging Tor: a formal treatment of onion encryption. *Eurocrypt*, 2018.

[58] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[59] C. Dwork. Differential privacy. *Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[60] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. *Eurocrypt*, 2006.

[61] M. Edman and B. Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM CSUR*, 2009.

[62] N. S. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on tor using long paths. *USENIX Security*, 2009.

[63] P. Fauzi et al. Quisquis: A new design for anonymous cryptocurrencies. *AsiaCrypt*, 2019.

[64] J. Feigenbaum, A. Johnson, and P. Syverson. A model of onion routing with provable anonymity. *Financial Cryptography and Data Security*, 2007.

[65] J. Feigenbaum, A. Johnson, and P. Syverson. Anonymity analysis of onion routing in the universally composable framework. *Workshop on Provable Privacy*, 2012.

[66] J. Feigenbaum, A. Johnson, and P. Syverson. Probabilistic analysis of onion routing in a black-box model. *ACM TISSEC*, 2012.

[67] E. Foundation. Ethereum Project. https://www.ethereum.org/, 2021. [Online; last accessed May-2021].

[68] G. Fuchsbauer. Subversion-zero-knowledge SNARKs. *PKC*, 2018.

[69] R. Fugger. Money as ious in social trust networks & a proposal for a decentralized currency network protocol. http://ripple.sourceforge.net, 2004. [Online; last accessed May-2021].

[70] A. Fujioka, Y. Okamoto, and T. Saito. Security of sequential multiple encryption. *Latincrypt*, 2010.

[71] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 1985.

[72] N. Gelernter and A. Herzberg. On the limits of provable anonymity. *WPES*, 2013.

[73] D. Genkin, D. Papadopoulos, and C. Papamanthou. Privacy in decentralized cryptocurrencies. *Commun. ACM*, 2018.

[74] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. *workshop on information hiding*, 1996.

[75] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of computer and system sciences*, 1984.

[76] M. Green and I. Miers. Bolt: Anonymous payment channels for decentralized currencies. *ACM CCS*, 2017.

[77] J. Groth. On the size of pairing-based non-interactive arguments. *Eurocrypt*, 2016.

[78] J. Groth and M. Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. *Crypto*, 2017.

[79] J. Y. Halpern and K. R. O'Neill. Anonymity and Information Hiding in Multiagent Systems. *Journal of Computer Security*, 2005.

[80] H. Halpin and M. Piekarska. Introduction to security and privacy on the blockchain. *EuroS&P Workshops*, 2017.

[81] E. Heilman et al. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. *NDSS*, 2017.

[82] A. Hevia and D. Micciancio. An indistinguishability-based characterization of anonymous channels. *PETS*, 2008.

[83] D. J. D. Hughes and V. Shmatikov. Information hiding, anonymity and privacy: A modular approach. *J. Comput. Secur.*, 2004.

[84] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. An empirical analysis of anonymity in zcash. In *USENIX Security*, 2018.

[85] G. Kappos, H. Yousaf, A. Piotrowska, S. Kanjalkar, S. Delgado-Segura, A. Miller, and S. Meiklejohn. An empirical analysis of privacy in the lightning network. *FC*, 2021.

[86] G. Karame and S. Capkun. Blockchain security and privacy. *IEEE S&P*, 2018.

[87] A. Kate, G. M. Zaverucha, and I. Goldberg. Pairing-based onion routing with improved forward secrecy. *ACM TISSEC*, 2010.

[88] T. Kerber et al. Ouroboros crypsinous: Privacy-preserving proof-of-stake. *IEEE S& P*, 2019.

[89] M. Klooß, A. Lehmann, and A. Rupp. (R)CCA secure updatable encryption with integrity protection. *Eurocrypt*, 2019.

[90] A. E. Kosba et al. How to use snarks in universally composable protocols. ePrint, http://eprint.iacr.org/2015/1093, 2015.

[91] P. Koshy, D. Koshy, and P. McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. *FC*, 2014.

[92] C. Kuhn, M. Beck, S. Schiffner, E. Jorswieck, and T. Strufe. On privacy notions in anonymous communication. In *arXiv, https://arxiv.org/abs/1812.05638*, 2018.

[93] C. Kuhn, M. Beck, S. Schiffner, E. Jorswieck, and T. Strufe. On privacy notion in anonymous communication. *PETS*, 2019.

[94] C. Kuhn, M. Beck, and T. Strufe. Breaking and (Partially) Fixing Provably Secure Onion Routing. *arXiv, https://arxiv.org/abs/1910.13772*, 2019.

[95] C. Kuhn, M. Beck, and T. Strufe. Breaking and (Partially) Fixing Provably Secure Onion Routing. *IEEE S&P*, 2020.

[96] C. Kuhn, M. Beck, and T. Strufe. Covid notions: Towards formal definitions–and documented understanding–of privacy goals and claimed protection in proximity-tracing services. *Elsevier Journal: Online Social Networks and Media*, 2021.

[97] C. Kuhn, D. Hofheinz, A. Rupp, and T. Strufe. Onion routing with replies. *Asiacrypt*, 2021.

[98] C. Kuhn, D. Hofheinz, A. Rupp, and T. Strufe. Onion routing with replies. *eprint, https://eprint.iacr.org/2021/1178*, 2021.

[99] C. Kuhn, F. Kitzing, and T. Strufe. Sok on performance bounds in anonymous communication. *WPES*, 2020.

[100] C. Kuhn, M. Noppel, C. Wressnegger, and T. Strufe. Plausible deniability for anonymous communication. *WPES*, 2021.

[101] A. Kumar et al. A traceability analysis of monero's blockchain. *ESORICS*, 2017.

[102] K. Kurosawa and Y. Desmedt. A new paradigm of hybrid encryption scheme. *crypto*, 2004.

[103] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle. *PETS*, 2016.

[104] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A scalable, traffic analysis resistant anonymity network for voip systems. *ACM SIGCOMM*, 2015.

[105] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. *Data Engineering*, 2007.

[106] P.-H. Lin, C. Kuhn, T. Strufe, and E. A. Jorswieck. Physical layer privacy in broadcast channels. *IEEE Workshop on Information Forensics and Security (WIFS)*, 2019.

[107] H. Lipmaa. Simulation-extractable SNARKs revisited. eprint, https://eprint.iacr.org/2019/612, 2019.

[108] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2007.

[109] G. Malavolta et al. Concurrency and privacy with payment-channel networks. *ACM CCS*, 2017.

[110] G. Malavolta et al. Silentwhispers: Enforcing security and privacy in decentralized credit networks. *NDSS*, 2017.

[111] S. Mauw, J. H. Verschuren, and E. P. de Vink. A formalization of anonymity and onion routing. *ESORICS*, 2004.

[112] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: Characterizing payments among men with no names. *IMC*, 2013.

[113] S. Micali. CS proofs (extended abstracts). *FOCS*, 1994.

[114] P. Moreno-Sanchez et al. Privacy preserving payments in credit networks. *NDSS*, 2015.

[115] P. Moreno-Sanchez, T. Ruffing, and A. Kate. Pathshuffle: Credit mixing and anonymous payments for ripple. *PETS*, 2017.

[116] P. Moreno-Sanchez, M. B. Zafar, and A. Kate. Listening to whispers of ripple: Linking wallets and deanonymizing transactions in the ripple network. In *PoPETS*, 2016.

[117] M. Möser and R. Böhme. The price of anonymity: empirical evidence from a market for bitcoin anonymization. *Journal of Cybersecurity*, 2017.

[118] M. Möser et al. An empirical analysis of traceability in the monero blockchain. Technical Report 1704.04299, arXiv, 2017.

[119] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. *IEEE S&P*, 2005.

[120] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[121] A. Narayanan, E. Shi, and B. I. P. Rubinstein. Link prediction by de-anonymization: How We Won the Kaggle Social Network Challenge. *Joint Conference on Neural Networks*, 2011.

[122] Nym—building the next generation of privacy infrastructure. `https://nymtech.net/`, 2020. [Online; last accessed May-2021].

[123] S. Oya, C. Troncoso, and F. Pérez-González. Do dummies pay off? limits of dummy traffic protection in anonymous communications. *PETS*, 2014.

[124] K. Peng. A general and efficient countermeasure to relation attacks in mix-based e-voting. *Int. J. Inf. Secur.*, 2011.

[125] PePP-PT e.V. i.Gr. Pepp-pt ntk high-level overview. Technical report, 2020. https://github.com/pepp-pt/pepp-pt-documentation/blob/master/PEPP-PT-high-level-overview.pdf [Online; last accessed April-2020].

[126] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. 2010.

[127] A. Pfitzmann and M. Waidner. Networks without user observability. *Computers & Security*, 1987.

[128] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The loopix anonymity system. *USENIX Security*, 2017.

[129] D. J. Pohly and P. McDaniel. Modeling Privacy and Tradeoffs in Multichannel Secret Sharing Protocols. *IEEE/IFIP DSN*, 2016.

[130] P. H. Potgieter. An introduction to new media for South African students. 2009.

[131] PRIVATICS team Inria and Fraunhofer AISEC. Robust and privacy-preserving proximity tracing protocol. Technical report, 2020. https://github.com/ROBERT-proximity-tracing/documents [Online; last accessed April-2020].

[132] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. *Security and Privacy in Social Networks*, 2013.

[133] J. Ren and J. Wu. Survey on anonymous communications in computer networks. *Computer Communications*, 2010.

[134] M. Romiti et al. Cross-layer deanonymization methods in the lightning protocol. *FC*, 2021.

[135] D. Ron and A. Shamir. Quantitative Analysis of the Full Bitcoin Transaction Graph. *FC*, 2013.

[136] S. Roos and P. Moreno-Sanchez. Ripple Data Set of Speedy Murmurs. https://crysp.uwaterloo.ca/software/speedymurmurs/download.php/, 2021. [Online; last accessed May-2021].

[137] T. Ruffing, P. Moreno-Sanchez, and A. Kate. P2p mixing and unlinkable bitcoin transactions. *NDSS*, 2017.

[138] E. B. Sasson et al. Zerocash: Decentralized anonymous payments from bitcoin. *IEEE S&P*, 2014.

[139] S. A. Schneider and A. Sidiropoulos. CSP and anonymity. *ESORICS*, 1996.

[140] A. Serjantov, R. Dingledine, and P. Syverson. From a trickle to a flood: Active attacks on several mix types. *Information Hiding*, 2002.

[141] Secure hash algorithm-3 (sha-3). National Institute of Standards and Technology (NIST), FIPS PUB 202, U.S. Department of Commerce, 2015.

[142] E. Shimshock, M. Staats, and N. Hopper. Breaking and provably fixing minx. *PETS*, 2008.

[143] F. Shirazi, M. Simeonovski, M. R. Asghar, M. Backes, and C. Diaz. A survey on routing in anonymous communication protocols. *ACM CSUR*, 2018.

[144] D. S. Sidhu. The chilling effect of government surveillance programs on the use of the internet by muslim-americans. *U. Md. LJ Race, Religion, Gender & Class*, 2007.

[145] M. Spagnuolo, F. Maggi, and S. Zanero. "bitiodine: Extracting intelligence from the bitcoin network". *FC*, 2014.

[146] C. Stachl, Q. Au, R. Schoedel, S. D. Gosling, G. M. Harari, D. Buschek, S. T. Völkel, T. Schuwerk, M. Oldemeier, T. Ullmann, et al. Predicting personality from patterns of behavior collected with smartphones. *Proceedings of the National Academy of Sciences*, 2020.

[147] S. Steinbrecher and S. Köpsell. Modelling unlinkability. *Privacy Enhancing Technologies Workshop (PET)*, 2003.

[148] H. Sun. The capacity of anonymous communications. *IEEE Transactions on Information Theory*, 2018.

[149] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2002.

[150] P. F. Syverson and S. G. Stubblebine. Group principals and the formalization of anonymity. *Formal Methods*, 1999.

[151] F. Tramèr, D. Boneh, and K. Paterson. Remote side-channel attacks on anonymous transactions. In *USENIX Security*, 2020.

[152] C. Troncoso et al. Decentralized privacy-preserving proximity tracing. https://github.com/DP-3T/documents/blob/master/DP3T White Paper.pdf, Version: April 10, 2020. [Online; last accessed April-2020].

[153] F. Tschorsch. *Onions in the Queue: An Integral Networking Perspective on Anonymous Communication Systems*. PhD thesis, Humboldt-Universität zu Berlin, 2016.

[154] J. Van Den Hooff et al. Vuvuzela: Scalable private messaging resistant to traffic analysis. *ACM SOSP*, 2015.

[155] S. Vaudenay. Analysis of DP3T. Technical Report 2020/399, eprint, https://eprint.iacr.org/2020/399, 2020.

[156] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. *USENIX Security*, 2012.

[157] R. Zhang, R. Xue, and L. Liu. Security and privacy on blockchain. *ACM Comput. Surv.*, 2019.

A. Details to Notions

A.1. Options for Notions

A.1.1 Protocol-dependent: Sessions

To model sessions, we set the auxiliary information of a communication to the session ID ($sess$): $aux = sess$. However, as the adversary can choose this auxiliary information, we need to ensure that the scenarios cannot be distinguished just because the session identifier is observed. Hence, we definine $sess$ to be a number in most communications. Only for the session notions, we require special session IDs that correspond to the current challenge Ψ and stage x in all challenge rows: $(x, Ch\Psi)$. In this way, they have to be the same in both scenarios and a concrete $sess$ is only used in one stage of one challenge.

The session identifier that is handed to the ACN protocol model is a random number that is generated by the challenger when a new $sess$ is seen. Hence, neither leaking (it is a random number) nor linking session identifiers (it will be picked new and statistically independent for every challenge and stage) will give the attacker an advantage.

We formalize this in the following definition, where we also use '_' to declare that this part of a tuple can be any value.[1]

Definition 38 (Sessions)**.** *Let x be the stage and $u_0^a, u_1^a, u_0'^a, u_1'^a$ be the senders and receivers of the first challenge row of this challenge Ψ and stage in instance $a \in \{0,1\}$. Property sess is met, iff for all $a \in \{0,1\}$:*

$$sess : \forall (r_0^a, r_1^a) \in \mathsf{CR}(\underline{r}_0^a, \underline{r}_1^a) : (r_0^a, r_1^a) = (u_0^a, u_0'^a, \_, (x, Ch\Psi)), (u_1^a, u_1'^a, \_, (x, Ch\Psi))$$

As not all protocols use sessions, we allow to add sessions as an option to the notion X abbreviated by X_s.

A.1.2 Adversary Model: Corruption

The different corruption options have implications on the challenger, when a corrupt query or a batch query arrives.

Check on corrupt queries. This check depends on whether the user corruption is adaptive, static, or not allowed at all. The default case for notion X is adaptive corruption, i.e. the adversary can corrupt honest users at any time. With static corruption X_{c^-}, the adversary has to corrupt a set of users before she sends her first batch. The third option, X_{c^0}, is that no user corruption is allowed. We denote the set of corrupted users as $\hat{U}$.

Definition 39 (Corruption: Check on Corrupt Query)**.** *Let $\hat{U}$ be the set of already corrupted users, u the user in the corrupt query and the bit* subsequent *be true iff at least one batch query*

[1]E.g. $\exists(u, m, \_) \in r$ will be true iff $\exists u' : \exists(u, m, u') \in r$.

happened. The following properties are met, iff:

$$corr_{static}: \text{subsequent} \implies u \in \hat{U} \quad corr_{no}: \bot \qquad corr_{adaptive}: \top$$

Check on batch queries. In reality for most ACNs the privacy goal can be broken for corrupted users, e.g. a corrupted sender has no unobservability. Therefore, we need to assure that the adversary cannot distinguish the scenario because the behavior of corrupted users differs. This is done by assuring equal behavior $corr_{c^e}$ or banning such users from communicating $corr_{c^{sr}}, corr_{c^s}, corr_{c^r}$.

Definition 40 (Corruption: Check on Batch Query)**.** *The following properties are met, iff for all $a \in \{0,1\}$:*

$$\begin{aligned}
corr_{c^{sr}} &: \forall (u,u',m,aux) \in \underline{r}_0^a \cup \underline{r}_1^a : u \notin \hat{U} \wedge u' \notin \hat{U}\\
corr_{c^{s}} &: \forall (u,u',m,aux) \in \underline{r}_0^a \cup \underline{r}_1^a : u \notin \hat{U}\\
corr_{c^{r}} &: \forall (u,u',m,aux) \in \underline{r}_0^a \cup \underline{r}_1^a : u' \notin \hat{U}\\
corr_{c^{e}} &: \forall \hat{u} \in \hat{U} : r_{0_i}^a = (\hat{u},\_,m,\_) \implies r_{1_i}^a = (\hat{u},\_,m,\_) \wedge r_{0_i}^a = (\_,\hat{u},m,\_) \implies r_{1_i}^a = (\_,\hat{u},m,\_)
\end{aligned}$$

Of course user corruption is not the only important part of an adversary model. Other adversarial capabilities can be adjusted with other parts of our framework (like the corruption of other parts of the ACN with protocol queries).

A.1.3 Easier Analysis: Quantification

Challenge Complexity. Example: *Consider Alice using a protocol, that achieves $S\overline{O}$ for one challenge row ($S\overline{O}_{\mathsf{CR}_1}$), but not for two ($S\overline{O}_{\mathsf{CR}_2}$). This means in the case that Alice only communicates once, the adversary is not able to distinguish Alice from any other potential sender Charlie. However, if Alice communicates twice the regime might distinguish her existence from the existence of some other user, e.g. by using an intersection attack.*

To quantify how different the scenarios can be, we add the concept of *challenge complexity*. Challenge complexity is measured in *Challenge rows*, the pairs of communications that differ in the two scenarios as defined earlier. c is the maximal allowed number of challenge rows in the game. Additionally, we add the maximal allowed numbers of challenge rows per challenge $\#cr$ as option to a notion X with $X_{\mathsf{CR}_{\#cr}}$.
Definition 41 (Challenge Complexity)**.** *Let $\#\mathsf{CR}$ be the number of challenge rows in this challenge so far. We say that the following property is met, iff:*

$$\mathsf{CR}_{\#cr}: \#\mathsf{CR} \leq \#cr$$

Challenges Cardinality. So far, our definitions focused on one challenge. We now bound the number of challenges to n, as the adversary potentially gains more information the more challenges are played. While challenge complexity defines a bound on the total number of differing rows within a single challenge, cardinality bounds the total number of challenges. Communications belonging to a challenge are identified by the challenge number Ψ, which has to be between 1 and n to be valid. The challenge number is a part of the auxiliary information of the communication and is only used by the challenger, not by the protocol model.

This dimension of quantification can be useful for analysis, since for certain assumptions the privacy of the n-challenge-case can be bounded in the privacy of the single-challenge-case as we will discuss in the next section.

Table A.1: Definition of notions including the options; for all notions X

Notion including option	**Definition**
X_s	Properties of $X \wedge sess$
X_{c^e}, $X_{c^{\mathrm{sr}}}$ etc.	$\wedge corr_{c^e}$, $\wedge corr_{c^{\mathrm{sr}}}$ etc.
$X_{CR_{\#cr}}$	Properties of $X \wedge \mathsf{CR}_{\#cr}$

A.1.4 Capturing Different Adversaries

Adversary Classes. Adversary classes are PPT algorithms that can modify and filter all in- and outputs from/to the adversary. Adversary classes $\mathcal{C}$ can be included into our framework in exactly the same way: to wrap the adversary $\mathcal{A}$. Instead of sending the queries to Ch, $\mathcal{A}$ sends the queries to $\mathcal{C}$, which can modify and filter them before sending them to Ch. Similarly, the answers from Ch are sent to $\mathcal{C}$ and possibly modified and filtered before sent further to $\mathcal{A}$. Adversary classes that fulfill reliability, alpha-renaming and simulatability (see [11] for definitions) are called single-challenge reducible. For such adversary classes it holds that every protocol Π that is (c, ϵ, δ)-X for $\mathcal{C}(\mathcal{A})$, is also $(n \cdot c, n \cdot \epsilon, n \cdot \epsilon^{n\epsilon}\delta)$-X for $\mathcal{C}(\mathcal{A})$. Even though our framework extends AnoA's in multiple ways, the proof for multi-challenge generalization of AnoA is independent from those extensions and still applies to our framework (see Appendix A.5.3).

Note that since the adversary class $\mathcal{C}$ is only a PPT algorithm, $\mathcal{C}(\mathcal{A})$ still is a PPT algorithm and hence, a possible adversary against X when analyzed without an adversary class. So, while adversary classes can help to restrict the capabilities of the adversary, results shown in the case without an adversary class carry over.

A.2. Challenger

This section describes the queries to the challenger $Ch(\Pi, X, c, n, b)$. Pseudocode of our challenger is shown in Appendix A.3.

Batch Query. The batches $\underline{r}_0, \underline{r}_1$ that the adversary chooses for the two scenarios are represented in batch queries. When the challenger receives a batch query, it first checks if their challenge number Ψ is valid, i.e. $\Psi \in \{1, \dots, n\}$. Further, the challenger validates the communications that would be input to Π for $b = 0$ and $b = 1$ as explained below. If the game is not aborted so far, the challenger retrieves or creates the current state s of the challenge Ψ, which stores information to calculate the properties. Afterwards it checks if the allowed total number of challenge rows c is met. If all criteria are met so far, it checks that the properties of the privacy notion X are met by using the current state of the challenge s, the set of corrupted users $\hat{U}$, the instances for both scenarios $\underline{r_0}^a, \underline{r^a}_1, a \in \{0, 1\}$. Finally, it runs the instance belonging to the challenge bit b of this game and the for this challenge randomly chosen instance bit a, if the properties are matched. Otherwise, it returns $\perp$ and aborts the experiment. Running the scenario in the ACN protocol returns information that is forwarded to the adversary (or adversary class). This information is what an adversary is assumed to be able to observe.

Corrupt Query. Corrupt queries represent adaptive, momentary corruption of users (senders or receivers). If the corrupt query is valid, the challenger forwards it to the ACN protocol. The ACN protocol returns the current state of the user to the challenger, who forwards it to the adversary. Active attacks based on corruption are realized with protocol queries if the protocol model allows for them.

Protocol Query. Protocol queries allow the adversary e.g. to compromise parts of the network (not the users), set parameters of the ACN protocol or use other functionalities modeled in the protocol model, like e.g. active attacks. The meaning and validity of those queries is specific to the analyzed ACN protocol.

Switch Stage Query. If this query occurs and it is allowed, i.e. the notion contains a relevant property, the stage is changed from 1 to 2.

Validate Communications. If the analyzed ACN protocol specifies restrictions of senders and receiver-message pairs, their validity is checked by this function.

Run Protocol. Run protocol first creates a new random session identifier if there is not already one for the adversary chosen session with the extension ID. This is done to ensure that the ACN protocol is not broken only because the session identifier is leaked. Afterwards it passes the communications to the ACN protocol formalization.

Remark to simple properties and instances. In case the notion only uses simple properties, the challenger will pick $a = 0$ and check the properties for $\underline{r_1}_j = \underline{r_1^0}_j$ and $\underline{r_0}_j = \underline{r_0^0}_j$. In case the notion uses a combination of simple and complex properties, the challenger will check the simple properties for any pair $\underline{r_1}_j = \underline{r_1^a}_j$ and $\underline{r_0}_j = \underline{r_0^{a'}}_j$ resulting by any $a, a' \in \{0, 1\}$.

A.3. Notions in Pseudocode

Algorithms 1 and 2 describe the challenger and thereby also the notions and properties in detail.

CalcNewState always calculates the states for all user roles (senders and receivers). This is for improved readability. It would be sufficient to calculate the parts of the state needed for the current notion.

For the simple properties $checkFor$ uses $s_{b_k}^0$ from s_{sender} resp. $s_{b_k}'^0$ from s_{rec} to calculate U_b, Q_b, P_b and H_b and compares them like in Definition 8. For the complex properties the senders and receivers of the first challenge row are stored in the $users$-part and the current stage in the $stage$-part of s. With this complex properties are computed as stated in Definition 10. Further, for the sessions-aspect the $session$-part of the state is set to $\perp$ if another sender-receiver-pair is used. With this and the $users-$ and $stage$-information the Definition 38 can be checked. For the corruption it gets all the required information direct as input and can check it like defined in Definition 40. For the challenge complexity the number of challenge rows of this challenge is counted in the cr-part of the state and hence, Definition 41 can be calculated.

A.4. Additional Tables and Figure

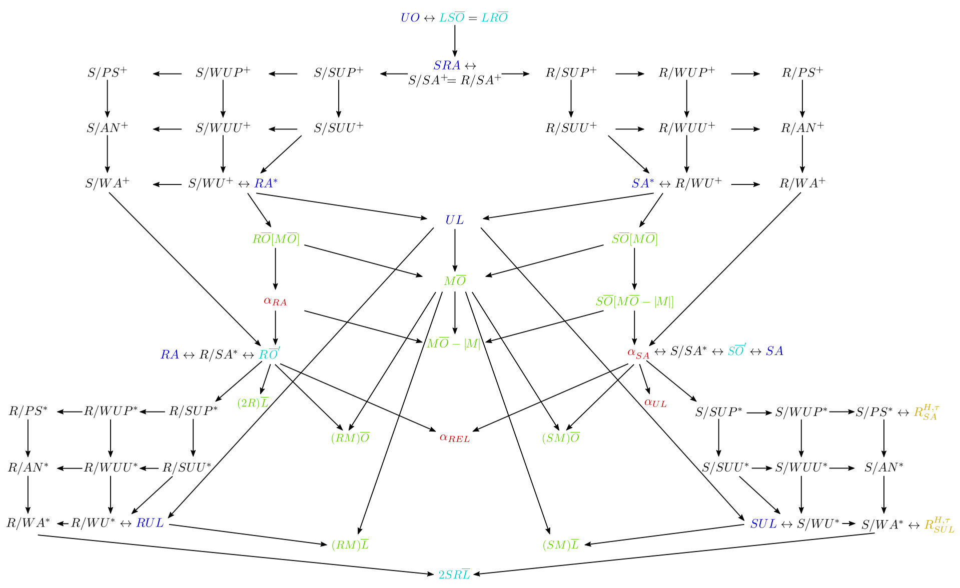


Figure A.1: Our hierarchy with the mapping to other works (Bohli's, AnoA, Hevia's, Gelernter's framework, Loopix's ACN and new notions)

Algorithm 1: Challenger $Ch(\Pi, X, c, n, b)$

```
$\hat{U} = \emptyset$
stage = 1
Upon query (Batch, $\underline{r}_0^0, \underline{r}_0^1, \underline{r}_1^0, \underline{r}_1^1, \Psi$)
    if $\Psi \notin \{1, \dots, n\}$ then
        output $\perp$
    if $\Psi \in T$ then
        Retrieve $s := s_\Psi$
    else
        $s := initializeState$
        if $X$ uses only simple properties then
            $a \leftarrow 0$
        else
            $a \leftarrow^R \{0,1\}$
        add $\Psi$ to $T$
    if $\neg Validate(r)$ then
        output $\perp$
    Compute $c_t = c_t + |CR(\underline{r}_0^0, \underline{r}_0^1, \underline{r}_1^0, \underline{r}_1^1)|$
    if $c_t > c$ then
        output $\perp$
    $s' = calculateNewState(stage, s, \underline{r}_0^0, \underline{r}_0^1, \underline{r}_1^0, \underline{r}_1^1))$
    if $checkFor(X, \Psi, s', \hat{U}, \underline{r}_0 = (\underline{r}_0^0, \underline{r}_0^1), \underline{r}_1 = (\underline{r}_1^0, \underline{r}_1^1))$ then
        $(\underline{r}, s_\Psi) \leftarrow (\underline{r}_b^a, s')$
    else
        output $\perp$
    Store $s_\Psi$
    RunProtocol $(\underline{r})$
Upon query (Protocol, x)
    if $x$ allowed then
        Send $x$ to $\Pi$
Upon query (Corrupt, u)
    if $X = X'_{c^0}$ ($X' \in$ Privacy notions) then
        output $\perp$
    if $X = X'_{c^-}$ ($X' \in$ Privacy notions) and a batch query occurred before and $u \notin \hat{U}$ then
        output $\perp$
    $\hat{U} = \hat{U} \cup \{u\}$
    Send internal state of $u$ to $\mathcal{A}$
Upon query (SwitchStage)
    if $\neg X$ includes $T_S$ or $T_R$ then
        output $\perp$
    stage = 2
Validate ($\underline{r}_0^0 = (S_{0_i}^0, R_{0_i}^0, m_{0_i}^0, aux_{0_i}^0)_{i \in \{1,\dots,l\}}, \underline{r}_0^1, \underline{r}_1^0, \underline{r}_1^1$)
    for $r = (S, R, m, aux) \in \{\underline{r}_{b'}^{a'} \mid a', b' \in \{0,1\}\}$ do
        if $\neg Validate(S, R, m)$ then
            output FALSE
    output TRUE
Run Protocol ($\underline{r} = (S_i, R_i, m_i, aux_i)_{i \in \{1,\dots,l\}}$)
    for $r_i \in \underline{r}$ do
        if $aux_i = (session_i, ID_i)$ then
            if $\nexists y : (session, y, ID_i) \in S_i$ then
                $y' \leftarrow \{0,1\}^k$
                Store (session, $y', ID$) in $S_i$
            else
                $y' := y$ from (session, $y, ID_i$) $\in S_i$
            Run $\Pi$ on $r_i$ with session ID $y'$
            Forward responses sent by $\Pi$ to $\mathcal{A}$
        else
            Run $\Pi$ on $r_i$
            Forward responses sent by $\Pi$ to $\mathcal{A}$
```

Algorithm 2: State Management

```
initializeState
    $s = (1, 1, (\tilde{s}, \tilde{s}, \tilde{s}, \tilde{s}, \tilde{r}, \tilde{r}, \tilde{r}, \tilde{r}), 0, \emptyset, \emptyset)$
    return $s$
calcNewState ($newStage, s = (stage, session, users, cr,\ s_{sender} = (L^0_{0_i}, L^1_{0_i}, L^0_{1_i}, L^1_{1_i})_{i \in \{1,\dots,k-1\}},$
 $s_{rec} = (L'^0_{0_i}, L'^1_{0_i}, L'^0_{1_i}, L'^1_{1_i})_{i \in \{1,\dots,k-1\}}),\ \underline{r}^0_0 = (S^0_{0_i}, R^0_{0_i}, m^0_{0_i}, aux^0_{0_i})_{i \in \{1,..,l\}},$
 $\underline{r}^1_0 = (S^1_{0_i}, R^1_{0_i}, m^1_{0_i}, aux^1_{0_i})_{i \in \{1,..,l\}},\ \underline{r}^0_1 = (S^0_{1_i}, R^0_{1_i}, m^0_{1_i}, aux^0_{1_i})_{i \in \{1,..,l\}}),$
 $\underline{r}^1_1 = (S^1_{1_i}, R^1_{1_i}, m^1_{1_i}, aux^1_{1_i})_{i \in \{1,..,l\}})$
    for $a \in \{0, 1\}$ do
        for $b \in \{0, 1\}$ do
            $L^a_{b_k} = \{(u, M) \mid M = \cup_{j: S^a_{b_j} = u} m^a_{b_i}\}$
            $L'^a_{b_k} = \{(u, M) \mid M = \cup_{j: R^a_{b_j} = u} m^a_{b_i}\}$
    $cr = cr + |CR(\underline{r}^0_0, \underline{r}^1_0, \underline{r}^0_1, \underline{r}^1_1)|$
    if $users = (\tilde{s}, \tilde{s}, \tilde{s}, \tilde{s}, \tilde{r}, \tilde{r}, \tilde{r}, \tilde{r}) \wedge cr > 0$ then
        $((S^0_0, R^0_0, \_, \_),\ (S^1_0, R^1_0, \_, \_), (S^0_1, R^0_, \_, \_),\ (S^1_1, R^1_1, \_, \_), \dots) = CR(\underline{r}^0_0, \underline{r}^1_0, \underline{r}^0_1, \underline{r}^1_1)$
        users= $(S^0_0, S^1_0, S^0_1, S^1_1, R^0_0, R^1_0, R^0_1, R^1_1)$
    if $users = (S^0_0, S^1_0, S^0_1, S^1_1, R^0_0, R^1_0, R^0_1, R^1_1)\ \wedge \exists (r^0_0, r^1_0, r^0_1, r^1_1) \in CR(\underline{r}^0_0, \underline{r}^1_0, \underline{r}^0_1, \underline{r}^1_1)$ :
     $(r^0_0, r^1_0, r^0_1, r^1_1) \neq ((S^0_0, R^0_0, \_, \_, \_), (S^1_0, R^1_0, \_, \_, \_),\ (S^0_1, R^0_1, \_, \_, \_), (S^1_1, R^1_1, \_, \_, \_))$ then
        session=$\perp$
    stage = newStage
    output $s$
```

Table A.2: Properties

Symbol	Description
U/U'	Who sends/receives is equal for both scenarios.
Q/Q'	Which sender/receiver sends/receives how often is equal for both scenarios.
H/H'	How many senders/receivers send/receive how often is equal for both scenarios.
P/P'	Which messages are sent/received from the same sender/receiver is equal for both scenarios.
$\|U\|/\|U'\|$	How many senders/receivers communicate is equal for both scenarios.
$\|M\|$	Messages in the two scenarios always have the same length.
E_S	Everything but the senders is identical in both scenarios.
E_R, E_M	analogous
E_{SM}	Everything but the senders and messages is identical in both scenarios.
E_{RM}, E_{SR}	analogous
$\aleph$	nothing will be checked; always true
$E_\Diamond$	If something is sent in both scenarios, the communication is the same.
$\not\Diamond$	In every communication something must be sent.
R_{SR}	Adversary picks two sender-receiver-pairs. One of the senders and one of the receivers is chosen randomly. For b=0 one of the adversary chosen sender-receiver pairs is drawn. For b=1 the sender is paired with the receiver of the other pair.
R_{SM}, R_{RM}	analogous
T_S	Adversary picks two senders. The other sender might send the second time (stage 2). For b=0 the same sender sends in both stages, for b=1 each sender sends in one of the stages.
T_R	analogous
M_{SR}	Adversary picks two sender-receiver-pairs. Sender-receiver-pairs might be mixed. For b=0 both adversary chosen sender-receiver-pairs communicate. For b=1 both mixed sender-receiver-pairs communicate.
M_{SM}, M_{RM}	analogous

Table A.3: Symbols used in the Game

Symbol	**Description**
$\mathcal{A}$	Adversary
Ch	Challenger
Π	ACN protocol model
$b \in \{0,1\}$	Challenge bit
$g \in \{0,1\}$	Adversary's guess
$\underline{r}_0 = (r_{0_1}, r_{0_2}, \dots, r_{0_l})$	Batch of communications
$r_{b_i} \in \{\Diamond, (u, u', m, aux)\}$	Communication
$\Diamond$	Nothing is communicated
(u, u', m, aux)	m is sent from u to u' with auxiliary information aux
$(\underline{r}_{0_1}, \dots, \underline{r}_{0_k})$	(First) Scenario
$\perp$	Abort game
$CR(\underline{r}_0, \underline{r}_1)$	Challenge rows of batches $\underline{r}_0, \underline{r}_1$
Ψ	Challenge Number
n	Number of challenges allowed
c	Number of challenge rows allowed in game
$\hat{U}$	Set of corrupted users
$\mathcal{U}$	Set of possible senders
$\mathcal{U}'$	Set of possible receivers

Table A.4: Notions and Restriction Options

Symbol	**Description**
$S\overline{O}\{R\overline{O} - \lvert U' \rvert\}$	Sender/Message Unobservability with Receiver Unobservability leaking User Number
$S\overline{O}\{R\overline{O} - H'\}$	Sender/Message Unobservability with Receiver Unobservability leaking Histogram
$S\overline{O}\{R\overline{O} - P'\}$	Sender/Message Unobservability with Receiver Unobservability leaking Pseudonym
$S\overline{O}\{RF\overline{L}\}$	Sender/Message Unobservability with Receiver-Frequency Unlinkability
$S\overline{O}\{RF\overline{L} - H'\}$	Sender/Message Unobservability with Receiver-Frequency Unlinkability leaking Histogram
$S\overline{O}\{RF\overline{L} - P'\}$	Sender/Message Unobservability with Receiver-Frequency Unlinkability leaking Pseudonym
$S\overline{O}\{RM\overline{L}\}$	Sender/Message Unobservability with Receiver-Message Unlinkability
$S\overline{O}\{RM\overline{L} - P'\}$	Sender/Message Unobservability with Receiver-Message Unlinkability leaking Pseudonym
$S\overline{O}$	Sender Unobservability
$S\overline{O} - \lvert U \rvert$	Sender Unobservability leaking User Number
$S\overline{O} - H$	Sender Unobservability leaking Histogram
$S\overline{O} - P$	Sender Unobservability leaking Pseudonym
$SF\overline{L}$	Sender-Frequency Unlinkability
$SF\overline{L} - H$	Sender-Frequency Unlinkability leaking Histogram
$SF\overline{L} - P$	Sender-Frequency Unlinkability leaking Pseudonym
$SM\overline{L}$	Sender-Message Unlinkability
$SM\overline{L} - P$	Sender-Message Unlinkability leaking Pseudonym
$S\overline{O}[M\overline{O} - \lvert M \rvert]$	Sender Unobservability with Message Unobservability leaking Message Length
$(2S)\overline{O}$	Twice Sender Unobservability
$(SM)\overline{O}$	Sender-Message Pair Unobservability
$(SM)\overline{L}$	Sender-Message Pair Unlinkability
$S\overline{O}'$	Restricted Sender Unobservability
Receiver notions	analogous
$C\overline{O}$	Communication Unobservability
$\overline{O}$	Unobservability
$(SR)\overline{O}$	Sender-Receiver Unobservability
$M\overline{O}[M\overline{L}]$	Message Unobservability with Message Unlinkability
$M\overline{O} - \lvert M \rvert$	Message Unobservability leaking Message Length
$(SR)\overline{L}$	Sender-Receiver Pair Unlinkability
X	notion, standard assumptions
X	adaptive corruption
X_{c^0}	No corruption of users is allowed.
X_{c^-}	Only static corruption of users is allowed.
$X_{c^{\mathrm{sr}}}$	Corrupted users are not allowed to be chosen as senders or receivers.
X_{c^s}	Corrupted users are not allowed to be senders.
X_{c^r}	Corrupted users are not allowed to be receivers.
X_{c^e}	Corrupted nodes send/receive identical messages in both scenarios.
X	Communication of corrupted users not restricted.
X_s	Sender and receiver of challenge rows stay the same for this challenge and stage.
X	Sessions are not restricted.
$X_{CR_{\#cr}}$	$\#cr$ communications in the two scenarios are allowed to differ per challenge.

A.5. Delayed Proofs

A.5.1 Advantage Definitions

Equivalence to Other Definitions. Notice, that Definition 2 is equivalent to

$$\begin{aligned}&(1)\ \Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 0)\rangle] \leq \Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 1)\rangle] + \delta \text{ and}\\ &(2)\ \Pr[1 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 1)\rangle] \leq \Pr[1 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 0)\rangle] + \delta.\end{aligned}$$

(1): $|Pr[0 \mid 0] - Pr[0 \mid 1]| \leq \delta$ for all $\mathcal{A} \iff (Pr[0 \mid 0] - Pr[0 \mid 1] \leq \delta$ for all $\mathcal{A}) \wedge (Pr[0 \mid 1] - Pr[0 \mid 0] \leq \delta$ for all $\mathcal{A})$. To every attack $\mathcal{A}$ with $Pr[0 \mid 1] - Pr[0 \mid 0] > \delta$, we can construct $\mathcal{A}'$ with $Pr[0 \mid 0] - Pr[0 \mid 1] > \delta$. Since the definition requires the inequality to hold for all attacks, this is enough to prove that (1) implies the original, the other way is trivial. This is how we construct it: Given attack $\mathcal{A}$, we construct $\mathcal{A}'$ by changing the batches of the first with the second scenario. Hence, $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 0)\rangle] = \Pr[0 = \langle \mathcal{A}' \mid Ch(\Pi, X, c, 1)\rangle]$ and $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 1)\rangle] = \Pr[0 = \langle \mathcal{A}' \mid Ch(\Pi, X, c, 0)\rangle]$.

(2): To every attack $\mathcal{A}$ breaking (1), we can construct one with the same winning probability in (2). Given attacker $\mathcal{A}$, we construct $\mathcal{A}'$ as the one that changes the batches of the first with the second scenario and inverts the output of $\mathcal{A}$. Hence, $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 0)\rangle] = \Pr[1 = \langle \mathcal{A}' \mid Ch(\Pi, X, c, 1)\rangle]$ and $\Pr[1 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 0)\rangle] = \Pr[0 = \langle \mathcal{A}' \mid Ch(\Pi, X, c, 1)\rangle]$. Since we can reverse this operations by applying them again, we can also translate in the other direction.

Differential Privacy based Definition. For some use cases, e.g. if the court of your jurisdiction requires that the sender of a critical content can be identified with a minimal probability of a certain threshold e.g. 70%, a non-negligible δ might be sufficient. Hence, we allow to specify the parameter of δ and extend it with the allowed number of challenge rows c to finally include the well-known concept of differential privacy (see Section 2.3.1) into the following definition similar to previous work [11]:
Definition 42 (Achieving $(c, \epsilon, \delta) - X$). *An ACN protocol Π is $(c, \epsilon, \delta) - X$ with $c > 0$, $\epsilon \geq 0$ and $0 \leq \delta \leq 1$, iff for all PPT algorithms $\mathcal{A}$:*

$$Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 0)\rangle] \leq e^{\epsilon} Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, X, c, 1)\rangle] + \delta.$$

Notice that ϵ describes how close the probabilities of guessing right and wrong have to be. This can be interpreted as the quality of privacy for this notion. While δ describes the probability with which the ϵ-quality can be violated. Hence, every ACN protocol will achieve $(0, 1) - X$ for any notion X, but this result does not guarantee anything, since with probability $\delta = 1$ the ϵ-quality is not met.

The first variant (Definition 2) can be expressed in terms of the second (Definition 42) as Π achieves X, iff Π is $(c, 0, \delta) - X$ for a negligible δ and any $c \geq 0$.

A.5.2 Notion Relations

Loopix

Loopix $LS\overline{O}$. $C\overline{O} \implies LS\overline{O}$: by definition.

$LS\overline{O} \implies C\overline{O}$: We construct a successful attack on $LS\overline{O}$ by creating two new challenge rows $(r_0, \Diamond)$ and $(\Diamond, r_1)$ for every challenge row (r_0, r_1) in the successful attack on $C\overline{O}$.

Loopix $S\overline{O}'$. $S\overline{O} \Rightarrow S\overline{O}'$: by definition.
$S\overline{O} \Leftarrow S\overline{O}'$: Given an attack $\mathcal{A}$ on $(c, 0, 2\delta) - S\overline{O}$, where both chosen scenarios use all users (otherwise it would be a valid attack on $S\overline{O}'$). Let $(r_{2_1}, \dots, r_{2_l})$ be the same batch as the second batch of $\mathcal{A}$ except that whenever one of the two senders of the first challenge row from the original scenarios is used, it is replaced with an arbitrary other sender (that was not used in the first challenge row of the original scenarios). Let $P(0|2)$ be the probability that $\mathcal{A}$ outputs 0, when the new batches are run; $P(0|0)$ when the first scenario of $\mathcal{A}$ is run and $P(0|1)$ when the second is run. In the worst case for the attacker $P(0|2) = \frac{P(0|0)+P(0|1)}{2}$ (otherwise we would replace the scenario b where $|P(0|2) - P(0|b)|$ is minimal with the new one and get better parameters in the following calculation). Since $\mathcal{A}$ is an attack on $(c, 0, 2\delta) - S\overline{O}$, $P(0|0) > P(0|1) + 2\delta$. Transposing and inserting the worst case for $P(0|2)$ leads to: $P(0|0) > 2P(0|2) - P(0|0) + 2\delta \iff P(0|0) > P(0|2) + \delta$. Hence, using $\mathcal{A}$ with the adapted scenario is an attack on $(c, 0, \delta) - S\overline{O}'$[2].

Gelernter's relaxed notions

$R^{H,\tau}_{SA} \Rightarrow S\overline{O} - P$: Every attack on $S\overline{O} - P$ is valid against $R^{H,\tau}_{SA}$: Since P is fulfilled, for every sender u_0 in the first scenario, there exists a sender $\tilde{u}_0$ in the second scenario sending the same messages. Hence, the permutation between senders of the first and second scenario exists.

$R^{H,\tau}_{SA} \Leftarrow S\overline{O} - P$: Every attack on $R^{H,\tau}_{SA}$ is valid against $S\overline{O} - P$: Since there exists a permutation between the senders of the first and second scenario sending the same messages, the partitions of messages sent by the same sender are equal in both scenarios, i.e. P is fulfilled.

$R^{H,\tau}_{S\overline{L}} \iff SM\overline{L} - P$: Q is required in both notions by definition. Arguing that P resp. G is fulfilled given the other attack is analogous to $R^{H,\tau}_{SA} \iff S\overline{O} - P$.

Hierarchy

⇢ The dashed, green implications hold, because of the following and analogous proofs:
Claim 1. *($M\overline{O}[M\overline{L}] \implies SM\overline{L}$) If protocol Π achieves $(c, \epsilon, \delta) - M\overline{O}[M\overline{L}]$, it achieves $(c, \epsilon, \delta) - SM\overline{L}$.*

Proof. Given a valid attack $\mathcal{A}$ on $SM\overline{L}$. We show that $\mathcal{A}$ is a valid attack on $M\overline{O}[M\overline{L}]$: Because of $SM\overline{L}$, Q is fulfilled. Because of E_S, the receivers of the communications input to the protocol are the same in both scenarios. Hence, every receiver receives the same number of messages, i.e. Q' is fulfilled. So, every attack against $(c, \epsilon, \delta) - SM\overline{L}$ is valid against $(c, \epsilon, \delta) - M\overline{O}[M\overline{L}]$.

Now, assume a protocol Π that achieves $(c, \epsilon, \delta) - M\overline{O}[M\overline{L}]$, but not $(c, \epsilon, \delta) - SM\overline{L}$. Because it does not achieve $(c, \epsilon, \delta) - SM\overline{L}$, there has to exist an successful attack $\mathcal{A}$ on $(c, \epsilon, \delta) - SM\overline{L}$, i.e.

$$\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, SM\overline{L}, c, 0) \rangle] > e^{\epsilon} \cdot \Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, SM\overline{L}, c, 1) \rangle] + \delta.$$

We know $\mathcal{A}$ is also valid against $(c, \epsilon, \delta) - M\overline{O}[M\overline{L}]$. Thus, it exists an attack, with

$$\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, M\overline{O}[M\overline{L}], c, 0) \rangle] > e^{\epsilon} \cdot \Pr[0 = \langle \mathcal{A} \mid Ch(\Pi, M\overline{O}[M\overline{L}], c, 1) \rangle] + \delta,$$

which contradicts the assumption that Π achieves $(c, \epsilon, \delta) - M\overline{O}[M\overline{L}]$. □

⋯▶ The dotted, yellow implications hold, because of the following and analogous proofs:
Claim 2. *($S\overline{O}\{RM\overline{L} - P'\} \implies S\overline{O}$) If protocol Π achieves $(c, \epsilon, \delta) - S\overline{O}\{RM\overline{L} - P'\}$, it achieves $(c, \epsilon, \delta) - S\overline{O}$.*

[2]An analogous argumentation works for $(c, \epsilon - \ln(0.5), \delta) - S\overline{O} \Leftarrow (c, \epsilon, \delta) - S\overline{O}'$.

Proof. Given a valid attack $\mathcal{A}$ on $S\overline{O}$. We show that $\mathcal{A}$ is a valid attack on $S\overline{O}\{RM\overline{L} - P'\}$: Because of E_S of $S\overline{O}$ the receiver-message pairs of the communications input to the protocol are the same in both scenarios. Hence, every receiver receives the same messages, i.e. Q' and P' are fulfilled. So, every attack against $(c, \epsilon, \delta) - S\overline{O}$ is valid against $(c, \epsilon, \delta) - S\overline{O}\{RM\overline{L} - P'\}$.

Now, the proof by contradiction is done analogous to the proof of Claim 1. $\square$

⟶ All dark blue implications $(c, \epsilon, \delta) - X_1 \Rightarrow (c, \epsilon, \delta) - X_2$ follow from the definition of the notions: Every valid attack against X_2 is valid against X_1. This holds because $U \Rightarrow |U|$, $H \Rightarrow |U|$, $Q \Rightarrow U$, $P \Rightarrow H$, $Q \Rightarrow H$, $\not\Diamond \Rightarrow \aleph$ and obviously for any properties A and B: $A \wedge B \Rightarrow A$ resp. $A \wedge B \Rightarrow B$.

⇢ The dotted, red implications $(c, \epsilon, \delta) - X_1 \Rightarrow (c, \epsilon, \delta) - X_2$ hold, because of the following and analogous proofs:

Claim 3. *($R\overline{O}\{SM\overline{L}\} \implies R\overline{O}[M\overline{O}]$) If protocol Π achieves $(c, \epsilon, \delta) - R\overline{O}\{SM\overline{L}\}$, it achieves $(c, \epsilon, \delta) - R\overline{O}[M\overline{O}]$.*

Proof. Given attack $\mathcal{A}_1$ on $(c, \epsilon, \delta)R\overline{O}[M\overline{O}]$. We show that $\mathcal{A}_1$ is valid against $(c, \epsilon, \delta) - R\overline{O}\{SM\overline{L}\}$: Sending nothing is not allowed in $R\overline{O}[M\overline{O}]$ and hence, will not happen ($\not\Diamond$) and because of E_{RM}, every sender sends equally often in both scenarios, i.e. Q is fulfilled.

Now, the proof by contradiction is done analogous to the proof of Claim 1. $\square$

⟶ The cyan implications $(c, \epsilon, \delta) - X_1 \Rightarrow (c, \epsilon, 2\delta) - X_2$ hold, because of the following and analogous proofs [3]:

Claim 4. *($R\overline{O} \implies (SR)\overline{O}$) If protocol Π achieves $(c, \epsilon, \delta) - R\overline{O}$, it achieves $(c, \epsilon, 2\delta) - (SR)\overline{O}$.*

Proof. We first argue the case of one challenge and later on extend it to multiple challenges. Given attack $\mathcal{A}_2$ on $(1, \epsilon, 2\delta) - (SR)\overline{O}$. We construct two attacks $\mathcal{A}_1'$ and $\mathcal{A}_1''$ against $R\overline{O}$ and show that one of those has at least the desired success.

We construct attacks $\mathcal{A}_1'$ and $\mathcal{A}_1''$. We therefore pick $a' = 0$ and $a'' = 1$. Those shall replace a, which would be picked randomly by the challenger in $(SR)\overline{O}$ to determine the batch instance. In $\mathcal{A}_1'$ we use the communications of $\mathcal{A}_2$ corresponding to $a' = 0$ (for $b = 0$ and $b = 1$) as challenge row, whenever a batch in $\mathcal{A}_2$ includes a challenge row. In $\mathcal{A}_1''$ we analogously use the communications corresponding to $a'' = 1$.

We show that both $\mathcal{A}_1'$ and $\mathcal{A}_1''$ are valid against $(1, \epsilon, \delta) - R\overline{O}$: Sending nothing is also not allowed in $(SR)\overline{O}$ and hence, will not happen ($\not\Diamond$) and because of the fixed $a = a'$ or $a = a''$, the senders of challenge rows are the same in both scenarios. Since also messages are equal in $(SR)\overline{O}$, the sender-message pairs are fixed (E_R). Hence, $\mathcal{A}_1'$ and $\mathcal{A}_1''$ are valid against $R\overline{O}$.

Since $\mathcal{A}_2$ is an successful attack on $(1, \epsilon, 2\delta) - (SR)\overline{O}$ and $\mathcal{A}_1'$ and $\mathcal{A}_1''$ against $R\overline{O}$ only fix the otherwise randomly picked a:

$$\begin{aligned}&0.5\Pr[0 = \langle \mathcal{A}_1' \mid Ch(\Pi, R\overline{O}, c, 0)\rangle] + 0.5\Pr[0 = \langle \mathcal{A}_1'' \mid Ch(\Pi, R\overline{O}, c, 0)\rangle]\\ &>e^{\epsilon} \cdot (0.5\Pr[0 = \langle \mathcal{A}_1' \mid Ch(\Pi, R\overline{O}, c, 1)\rangle] + 0.5\Pr[0 = \langle \mathcal{A}_1'' \mid Ch(\Pi, R\overline{O}, c, 1)\rangle]) + 2\delta.\end{aligned}$$

So,

$$\begin{aligned}&0.5\Pr[0 = \langle \mathcal{A}_1' \mid Ch(\Pi, R\overline{O}, c, 0)\rangle] > e^{\epsilon} \cdot (0.5\Pr[0 = \langle \mathcal{A}_1' \mid Ch(\Pi, R\overline{O}, c, 1)\rangle] + \delta \text{ or}\\ &0.5\Pr[0 = \langle \mathcal{A}_1'' \mid Ch(\Pi, R\overline{O}, c, 0)\rangle] > e^{\epsilon} \cdot (0.5\Pr[0 = \langle \mathcal{A}_1'' \mid Ch(\Pi, R\overline{O}, c, 1)\rangle] + \delta\end{aligned}$$

[3]For $S\overline{O} \Rightarrow (SR)\overline{O}$ (resp. $S\overline{O} \Rightarrow (SM)\overline{O}$) pick challenge rows differently; for $b = 0 : a = a'$ and for $b = 1 : a = 1 - a'$ to ensure that receivers (resp. messages) are equal.
For $(2c, \epsilon, \delta) - SM\overline{L} \Rightarrow (c, \epsilon, \delta) - (SM)\overline{L}$, (resp. $RM\overline{L} \Rightarrow (RM)\overline{L}$, $SM\overline{L} - P \Rightarrow (SR)\overline{L}$) replace the challenge row with the corresponding two rows.

must hold true (otherwise we get a contradiction with the above inequality). Hence, $\mathcal{A}_1'$ or $\mathcal{A}_1''$ has to successfully break $(1, \epsilon, \delta) - R\overline{O}$.

In case of multiple challenges: the instance bit is picked randomly for every challenge. Hence, we need to construct one attack for every possible combination of instance bit picks, i.e. 2^c attacks in total[4] from which each is a PPT algorithm and at least one is at least as successful as the attack on $(c, \epsilon, 2\delta) - (SR)\overline{O}$. Now, the proof by contradiction is done analogous to the proof of Claim 1. □

Remark. c can be any value in the proofs (esp. $c = 1$ or $c > 1$) the proposed constructions apply changes for each challenge row.

A.5.3 Multi-Challenge Generalization

Proof. [Proof sketch] The proof is analogous to the one in Appendix A.1 of [11][5]: we only argue that our added concepts (adaptive corruption, arbitrary sessions and grouping of challenge and input queries to batches) do not change the indistinguishability of the introduced games.

Adaptive Corruption. In Games G_0 till G_2, G_3 till G_6 and G_9 to G_{10} communications that reach the protocol model are identical. Hence, also adaptive corruption queries between the batch queries will return the same results (if adaptive corruption is used probabilistically: the probability distribution for the results is equal in all these games). G_2 to G_3 and G_7 to G_8 by induction hypothesis. G_6 to G_7 and G_8 to G_9 adaptive corruption is independent from the used challenge numbers (called challenge tags in [11]).

The argumentation for sessions and batches in analogous. Notice that by using the batch concept, in some games a part of a batch might be simulated, while another part is not. □

A.5.4 UC-Realizability

Proof. [Proof sketch] AnoA's proof assumes that Π does not achieve $(\epsilon, \delta + \delta')$-X. Hence, there must be an attack $\mathcal{A}$ distinguishing the scenarios. With this, they build a PPT distinguisher $\mathcal{D}$ that uses $\mathcal{A}$ to distinguish Π from $\mathcal{F}$. Since, even with our extensions $\mathcal{A}$ still is a PPT algorithm, that can be used to build distinguisher $\mathcal{D}$ and the inequalities that have to be true are the same (since the definition of achieving (ϵ, δ)-X is the same as being (ϵ, δ)-differentially private). The combination of Π not being $(\epsilon, \delta + \delta')$-X and $\mathcal{F}$ being $(\epsilon, \delta) - X$ results in the same contradiction as in AnoA's proof. □

A.5.5 Completeness of Hierarchy

Proof. We give the systematic overview over all combinations from sender or impartial notions to all other notions and which proof applies for which relation in Table A.5. Since receiver notions are completely analogous to sender notions, we spare this part of the table.

In Table A.5 "⇒" indicates that the notion of the column implies the one of the row. "=" is used when the notions are equal. P_n indicates that the counterexample is described in a proof with number n. P_A and P_B are proofs covering multiple counterexamples and P_A^* is a special one of those counterexamples. P_n' means the proof analogous to P_n, but for the receiver notion. (P_n) means that there cannot be an implication, because otherwise it would be a contradiction with proof P_n since our implications are transitive.

[4]Note that this does not contradict the PPT requirement of our definition as only finding the right attack is theoretically exponential in the number of challenges allowed. However, the attack itself is still PPT (and might be even easier to find for a concrete protocol).

[5]Note, although they include the challenge number n in the definition of achieving a notion, this is not used in the theorem.

Table A.5: Completeness; proofs for all relations between the notions (Part 1/3)

	$C\overline{O}$	$\overline{O}$	$M\overline{O}[M\overline{L}]$	$M\overline{O}$	$M\overline{O}-\lvert M\rvert$	$(SR)\overline{O}$	$(SR)\overline{L}$	$S\overline{O}\{R\overline{O}-\lvert U'\rvert\}$	$S\overline{O}\{R\overline{O}-H'\}$	$S\overline{O}\{R\overline{O}-P'\}$	$S\overline{O}\{RF\overline{L}\}$	$S\overline{O}\{RF\overline{L}-H'\}$	$S\overline{O}\{RF\overline{L}-P'\}$	$S\overline{O}\{RM\overline{L}\}$	$S\overline{O}\{RM\overline{L}-P'\}$	$S\overline{O}[M\overline{O}]$	$S\overline{O}[M\overline{O}-\lvert M\rvert]$
$C\overline{O}$	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$\overline{O}$	P_1	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$M\overline{O}[M\overline{L}]$	(P_1)	(P_2)	$=$	$\Rightarrow$	$\Rightarrow$	P_2	$\Rightarrow$	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)
$M\overline{O}$	(P_1)	(P_3)	(P_3)	$=$	$\Rightarrow$	(P_2)	P_3	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)
$M\overline{O}-\lvert M\rvert$	(P_1)	(P_3)	(P_3)	(P_{17})	$=$	(P_2)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)
$(SR)\overline{O}$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	$=$	P_{16}	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})
$(SR)\overline{L}$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	(P_7)	$=$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8')	(P_8')
$S\overline{O}\{R\overline{O}-\lvert U'\rvert\}$	(P_1)	(P_4)	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{R\overline{O}-H'\}$	(P_1)	(P_4)	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	$=$	$\Rightarrow$	(P_A)	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{R\overline{O}-P'\}$	(P_1)	(P_6)	(P_6)	(P_6)	P_6	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	$=$	(P_A)	(P_A)	$\Rightarrow$	(P_A)	$\Rightarrow$	(P_6)	(P_6)
$S\overline{O}\{RF\overline{L}\}$	(P_1)	(P_4)	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{RF\overline{L}-H'\}$	(P_1)	(P_4)	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{RF\overline{L}-P'\}$	(P_1)	(P_6)	(P_6)	(P_6)	(P_6)	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	$=$	(P_A)	$\Rightarrow$	(P_6)	(P_6)
$S\overline{O}\{RM\overline{L}\}$	(P_1)	(P_4)	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{RM\overline{L}-P'\}$	(P_1)	(P_6)	(P_6)	(P_6)	(P_6)	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	$=$	(P_6)	(P_6)
$S\overline{O}[M\overline{O}]$	(P_1)	(P_5)	P_5	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	$=$	$\Rightarrow$
$S\overline{O}[M\overline{O}-\lvert M\rvert]$	(P_1)	(P_5)	(P_5)	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_{17})	$=$
$S\overline{O}$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	$\Rightarrow$	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_6)	(P_6)
$S\overline{O}-\lvert U\rvert$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	P_7	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_B)	(P_B)
$S\overline{O}-H$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	(P_7)	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_B)	(P_B)
$S\overline{O}-P$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	(P_7)	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_B)	(P_B)
$SF\overline{L}$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	(P_7)	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_B)	(P_B)
$SF\overline{L}-H$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	(P_7)	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_B)	(P_B)
$SF\overline{L}-P$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	(P_7)	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_B)	(P_B)
$SM\overline{L}$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	(P_7)	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_B)	(P_B)
$SM\overline{L}-P$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	(P_7)	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_B)	(P_B)
$(2S)\overline{O}$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	P_{10}	P_{11}	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})
$(SM)\overline{O}$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	P_{12}	P_{13}	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})
$(SM)\overline{L}$	(P_1)	(P_5)	(P_5)	(P_6)	(P_6)	P_{14}	P_{15}	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})

Table A.5: Completeness; proofs for all relations between the notions (Part 2/3)

	$S\overline{O}$	$S\overline{O}-\lvert U\rvert$	$S\overline{O}-H$	$S\overline{O}-P$	$SF\overline{L}$	$SF\overline{L}-H$	$SF\overline{L}-P$	$SM\overline{L}$	$SM\overline{L}-P$	$(2S)\overline{O}$	$(SM)\overline{O}$	$(SM)\overline{L}$	$R\overline{O}\{S\overline{O}-\lvert U\rvert\}$	$R\overline{O}\{S\overline{O}-H\}$	$R\overline{O}\{S\overline{O}-P\}$	$R\overline{O}\{SF\overline{L}\}$	$R\overline{O}\{SF\overline{L}-H\}$
$C\overline{O}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$\overline{O}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$M\overline{O}[M\overline{L}]$	(P_2)	$(P_A'^*)$	$(P_A'^*)$	$(P_A'^*)$	$(P_A'^*)$	$(P_A'^*)$	$(P_A'^*)$	$\Rightarrow$	$\Rightarrow$	(P_4')	$\Rightarrow$	$\Rightarrow$	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)
$M\overline{O}$	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_4')	$\Rightarrow$	$\Rightarrow$	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)
$M\overline{O}-\lvert M\rvert$	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_4')	P_{17}'	P_{18}'	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)
$(SR)\overline{O}$	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_4')	(P_{17}')	$P_{18}')$	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})
$(SR)\overline{L}$	(P_8')	(P_8')	(P_8')	(P_8')	(P_8')	(P_8')	(P_8')	(P_8')	(P_8')	(P_4')	(P_{19})	(P_{20})	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$S\overline{O}\{R\overline{O}-\lvert U'\rvert\}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)
$S\overline{O}\{R\overline{O}-H'\}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)
$S\overline{O}\{R\overline{O}-P'\}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)
$S\overline{O}\{RF\overline{L}\}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)
$S\overline{O}\{RF\overline{L}-H'\}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)
$S\overline{O}\{RF\overline{L}-P'\}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)
$S\overline{O}\{RM\overline{L}\}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)
$S\overline{O}\{RM\overline{L}-P'\}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)
$S\overline{O}[M\overline{O}]$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$S\overline{O}[M\overline{O}-\lvert M\rvert]$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$S\overline{O}$	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$S\overline{O}-\lvert U\rvert$	P_B	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_4')	P_{19}	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$S\overline{O}-H$	(P_B)	P_B	$=$	$\Rightarrow$		$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_4')	(P_{19})	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$S\overline{O}-P$	(P_B)	P_B	P_B	$=$	P_B	P_B	$\Rightarrow$	P_B	$\Rightarrow$	(P_4')	(P_{19})	P_{20}	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$SF\overline{L}$	(P_B)	P_B	P_B	P_B	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_4')	(P_{19})	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$SF\overline{L}-H$	(P_B)	P_B	P_B	P_B	P_B	$=$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_4')	(P_{19})	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$SF\overline{L}-P$	(P_B)	P_B	P_B	P_B	P_B	P_B	$=$	P_B	$\Rightarrow$	(P_4')	(P_{19})	(P_{20})	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$SM\overline{L}$	(P_B)	P_B	P_B	P_B	P_B	P_B	P_B	$=$	$\Rightarrow$	(P_4')	(P_{19})	$\Rightarrow$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$SM\overline{L}-P$	(P_B)	P_B	P_B	P_B	P_B	P_B	P_B	P_B	$=$	(P_4')	(P_{19})	(P_{20})	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)
$(2S)\overline{O}$	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	$=$	P_{21}	P_{22}	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})
$(SM)\overline{O}$	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_4')	$=$	P_{23}	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})
$(SM)\overline{L}$	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_4')	(P_{19})	$=$	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})

Table A.5: Completeness; proofs for all relations between the notions (Part 3/3)

	$R\overline{O}\{SF\overline{L}-P\}$	$R\overline{O}\{SM\overline{L}\}$	$R\overline{O}\{SM\overline{L}-P\}$	$R\overline{O}[M\overline{O}]$	$R\overline{O}[M\overline{O}-\|M\|]$	$R\overline{O}$	$R\overline{O}-\|U'\|$	$R\overline{O}-H'$	$R\overline{O}-P'$	$RF\overline{L}$	$RF\overline{L}-H'$	$RF\overline{L}-P'$	$RM\overline{L}$	$RM\overline{L}-P'$	$(2R)\overline{O}$	$(RM)\overline{O}$	$(RM)\overline{L}$
$C\overline{O}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$\overline{O}$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$
$M\overline{O}[M\overline{L}]$	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)	(P_2)	(P_A^*)	(P_A^*)	(P_A^*)	(P_A^*)	(P_A^*)	(P_A^*)	$\Rightarrow$	$\Rightarrow$	(P_4)	$\Rightarrow$	$\Rightarrow$
$M\overline{O}$	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_4)	$\Rightarrow$	$\Rightarrow$
$M\overline{O}-\|M\|$	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_3)	(P_4)	(P_{17})	(P_{18})
$(SR)\overline{O}$	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_{16})	(P_4)	(P_{17})	(P_{18})
$(SR)\overline{L}$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$S\overline{O}\{R\overline{O}-\|U'\|\}$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	P_A	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	P_4	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{R\overline{O}-H'\}$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	P_A	P_A	$\Rightarrow$	$\Rightarrow$	P_A	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_4)	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{R\overline{O}-P'\}$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	P_A	P_A	P_A	$\Rightarrow$	P_A	P_A	$\Rightarrow$	P_A	$\Rightarrow$	(P_4)	P_{24}	P_9
$S\overline{O}\{RF\overline{L}\}$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	P_A	P_A	P_A	P_A	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_4)	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{RF\overline{L}-H'\}$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	P_A	P_A	P_A	P_A	P_A	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	(P_4)	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{RF\overline{L}-P'\}$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	P_A	P_A	P_A	P_A	P_A	P_A	$\Rightarrow$	P_A	$\Rightarrow$	(P_4)	(P_{24})	(P_9)
$S\overline{O}\{RM\overline{L}\}$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	P_A	P_A	P_A	P_A	P_A	P_A	P_A^*	$\Rightarrow$	$\Rightarrow$	(P_4)	$\Rightarrow$	$\Rightarrow$
$S\overline{O}\{RM\overline{L}-P'\}$	(P_A)	(P_A)	(P_A)	(P_A)	(P_A)	P_A	P_A	P_A	P_A	P_A	P_A	P_A	P_A	$\Rightarrow$	(P_4)	(P_{24})	(P_9)
$S\overline{O}[M\overline{O}]$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	P_8	(P_4)	$\Rightarrow$	$\Rightarrow$
$S\overline{O}[M\overline{O}-\|M\|]$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	P_{17}	P_{18}
$S\overline{O}$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$S\overline{O}-\|U\|$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$S\overline{O}-H$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$S\overline{O}-P$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$SF\overline{L}$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$SF\overline{L}-H$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$SF\overline{L}-P$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$SM\overline{L}$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$SM\overline{L}-P$	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_8)	(P_4)	(P_{17})	(P_{18})
$(2S)\overline{O}$	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_{11})	(P_4)	(P_{17})	(P_{18})
$(SM)\overline{O}$	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_{13})	(P_4)	(P_{17})	(P_{18})
$(SM)\overline{L}$	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_{15})	(P_4)	(P_{17})	(P_{18})

Numbered Proofs. The proofs identified in Table A.5 follow.

Lemma 1. *P_4. $(c, \epsilon, \delta) - S\overline{O}\{R\overline{O} - |U'|\} \not\Rightarrow (c^*, \epsilon^*, \delta^*) - (2R)\overline{O}$ for any $\epsilon^* \geq 0, \delta^* < 1, c^* \geq 2$ and any $\epsilon \geq 0, \delta < 1, c \geq 1$.*

Proof. Given a protocol Π, that achieves $(c, \epsilon, \delta) - S\overline{O}\{R\overline{O} - |U'|\}$. Let protocol Π' be the protocol, that behaves like Π and additionally publishes the current number of receivers $|U'|$ after every batch. Since in $S\overline{O}\{R\overline{O} - |U'|\}$ the number of receivers always needs to be identical in both scenarios, outputting it will not lead to new information for the adversary. So, Π' achieves $(c, \epsilon, \delta) - S\overline{O}\{R\overline{O} - |U'|\}$.

Fix $\epsilon^* \geq 0, \delta^* < 1, c^* \geq 2$ arbitrarily. Let u'_0, u'_1 be valid receivers and m_0 a valid message. $(c^*, \epsilon^*, \delta^*) - (2R)\overline{O}$ of Π' can be broken by the following attack: An adversary $\mathcal{A}$ inputs a batch query with two valid challenge rows with differing receivers[6]: $((u'_0, m_0), \text{SwitchStageQuery}, (u'_0, m_0))$ as instance 0 of the first scenario, $((u'_1, m_0), \text{SwitchStageQuery}, (u'_1, m_0))$ as instance 1 of the first scenario and $((u'_0, m_0), \text{SwitchStageQuery}, (u'_1, m_0))$ as instance 0 and $((u'_1, m_0), \text{SwitchStageQuery}, (u'_0, m_0))$ as instance 1 of the second scenario. If Π' outputs the number of receivers as being 1, both messages have been received by the same user and $\mathcal{A}$ outputs 0. Otherwise the number of receivers is 2 and the messages have been received by different users. It outputs 1 in this case and wins the game with certainty. Hence, $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', (2R)\overline{O}, c^*, 0)\rangle] = 1$ and $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', (2R)\overline{O}, c^*, 1)\rangle] = 0$ and thus $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', (2R)\overline{O}, c^*, 0)\rangle] > e^{\epsilon^*} \cdot \Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', (2R)\overline{O}, c^*, 1)\rangle] + \delta^*$ (since $1 > e^{\epsilon^*} \cdot 0 + \delta^*$).[7] □

Tables A.6 summarizes the ideas of proofs that work analogously to Lemma 1. $I = (S, m)$ means the sender(S)-message(m) pairs are published, i.e. it is leaked who sent which message. 1.(S,m) means that the first sender-message pair is revealed. $|m|$ without brackets means the set of all message lengths is published; $|U|$ the number of senders. The other abbreviations are used analogously. The attack is shortened to the format ⟨(communications of instance 0 scenario 0),(communications of instance 1 of scenario 0)⟩,⟨(communications of instance 0 scenario 1),(communications of instance 1 of scenario 1)⟩ (if both instances of the scenario are equal, we shorten to:⟨(communications of instance 0 scenario 0)⟩,⟨(communications of instance 0 scenario 1)⟩) and all not mentioned elements are equal in both scenarios. m_0, m_1, m_2, m_3 are messages with $|m_0| < |m_1|$, $|m_2| = |m_3|$ and $m_0 \neq m_1 \neq m_2 \neq m_3$; u_0, u_1, u_2 senders and u'_0, u'_1, u'_2 receivers.

Lemma 2. *P_{10}. $(c, \epsilon, \delta) - (2S)\overline{O} \not\Rightarrow (c^*, \epsilon^*, \delta^*) - (SR)\overline{L}$ for any $\epsilon^* \geq 0, \delta^* < 1, c^* \geq 2$ and for any $\epsilon \geq 0, \delta < 1, c \geq 2$.*

Proof. Given a protocol Π with $(c, \epsilon, \delta) - C\overline{O}$. Let Π' behave like Π and additionally publish the first sender-receiver-pair. Since Π does not leak any information, Π' does not leak any information except the first sender-receiver-pair. Hence, Π' has $(c, \epsilon, \delta) - (2S)\overline{O}$, because who sends the second time is concealed (otherwise $(c, \epsilon, \delta) - C\overline{O}$ of Π could be broken based on this, which would be a contradiction to our assumption.).

Fix $\epsilon^* \geq 0, \delta^* < 1, c^* \geq 2$ and let u_0, u_1 be valid senders and u'_0, u'_1 valid receivers. Then the following attack on $(c^*, \epsilon^*, \delta^*) - (SR)\overline{L}$ is possible: The adversary $\mathcal{A}$ creates a batch query containing only challenge rows: $((u_0, u'_0), (u_1, u'_1))$ as instance 0 of the first scenario, $((u_1, u'_1), (u_0, u'_0))$ as instance 1 of the first scenario and $((u_0, u'_1), (u_1, u'_0))$ as instance 0 of the second scenario, $((u_1, u'_0), (u_0, u'_1))$ as instance 1 of the second scenario. If the published first sender-receiver pair is u_1, u'_1 or u_0, u'_0 , $\mathcal{A}$ outputs 0. Otherwise it outputs 1. Obviously, the adversary wins the game with certainty with this strategy. Hence, $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', (SR)\overline{L}, c^*, 0)\rangle] = 1$ and $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', (SR)\overline{L}, c^*, 1)\rangle] = 0$ and thus $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', (SR)\overline{L}, c^*, 0)\rangle] > e^{\epsilon^*} \cdot \Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', (SR)\overline{L}, c^*, 1)\rangle] + \delta^*$ (since $1 > e^{\epsilon^*} \cdot 0 + \delta^*$). □

[6]For a simplified representation we only present the parts of the communication that differ in both scenarios and spare the senders of the communications.

[7]Notice, that any protocol would achieve $(c^*, \epsilon^*, \delta^*) - (2R)\overline{O}$ for $c^* = 1$ since no complete challenge is possible there.

Table A.6: Counter example construction idea with $X_1' = X_1$

P_n	X_1	X_2	I	attack
P_4	$S\overline{O}\{R\overline{O} - \lvert U'\rvert\}$	$(2R)\overline{O}$	$\lvert U'\rvert$	$\langle((u_0', m_0), \text{switchStage}, (u_0', m_0)), ((u_1', m_0), \text{switchStage}, (u_1', m_0))\rangle, \langle((u_0', m_0), \text{switchStage}, (u_1', m_0)), ((u_1', m_0), \text{switchStage}, (u_0', m_0))\rangle$
P_6	$S\overline{O}\{R\overline{O} - P'\}$	$M\overline{O} - \lvert M\rvert$	m	$((m_2)), ((m_3))$
P_{24}	$S\overline{O}\{R\overline{O} - P'\}$	$(RM)\overline{O}$	$\lvert U'\rvert, m$	$\langle((u_0', m_0), (u_0', m_2)), ((u_0', m_0), (u_1', m_3))\rangle, \langle((u_0', m_0), (u_0', m_3)), ((u_0', m_0), (u_1', m_2))\rangle$
P_9	$S\overline{O}\{R\overline{O} - P'\}$	$(RM)\overline{L}$	P'	$\langle((u_0', m_2), (u_0', m_0), (u_1', m_1)), ((u_0', m_2), (u_1', m_1), (u_0', m_0))\rangle, \langle((u_0', m_2), (u_0', m_1), (u_1', m_0)), ((u_0', m_2), (u_1', m_0), (u_0', m_1))\rangle$
P_5	$S\overline{O}[M\overline{O}]$	$M\overline{O}[M\overline{L}]$	$1.R$	$((u_0'), (u_1')), ((u_1'), (u_0'))$
P_8	$S\overline{O}[M\overline{O}]$	$RM\overline{L} - P'$	$1.R$	$((u_0'), (u_1')), ((u_1'), (u_0'))$
P_{17}	$S\overline{O}[M\overline{O} - \lvert M\rvert]$	$(RM)\overline{O}$	$\lvert U'\rvert, \lvert m\rvert$	$\langle((u_0', m_2), (u_0', m_0)), ((u_0', m_2), (u_1', m_1))\rangle \langle((u_0', m_2), (u_0', m_1)), ((u_0', m_2), (u_1', m_0))\rangle$
P_{18}	$S\overline{O}[M\overline{O} - \lvert M\rvert]$	$(RM)\overline{L}$	$(R, \lvert m\rvert)$	$\langle((u_0', m_0), (u_1', m_1)), ((u_1', m_1), (u_0', m_0))\rangle \langle((u_0', m_1), (u_1', m_0)), ((u_1', m_0), (u_0', m_1))\rangle$
P_7	$S\overline{O} - \lvert U\rvert$	$(SR)\overline{O}$	$\lvert U'\rvert, \lvert U\rvert$	$\langle((u_0, u_0'), (u_0, u_0')), ((u_0, u_0'), (u_1, u_1'))\rangle \langle((u_0, u_0'), (u_0, u_1')) ((u_0, u_0'), (u_1, u_0'))\rangle$
P_{19}	$S\overline{O} - \lvert U\rvert$	$(SM)\overline{O}$	$\lvert U\rvert, \lvert m\rvert$	$\langle((u_0, m_2), (u_0, m_0)), ((u_0, m_2), (u_1, m_1))\rangle \langle((u_0, m_2), (u_0, m_1)), ((u_0, m_2), (u_1, m_0))\rangle$
P_{20}	$S\overline{O} - P$	$(SM)\overline{L}$	P	$\langle((u_0, m_2), (u_0, m_0), (u_1, m_1)), ((u_0, m_2), (u_1, m_1), (u_0, m_0))\rangle, \langle((u_0, m_2), (u_0, m_1), (u_1, m_0)), ((u_0, m_2), (u_1, m_0), (u_0, m_1))\rangle$
P_n'	Receiver notions	analogous	analogous	
P_1	$\overline{O}$	$C\overline{O}$	$\aleph$	$((u_0)), (\Diamond)$
P_2	$M\overline{O}[M\overline{L}]$	$(SR)\overline{O}$	Q, Q'	$\langle((u_0, u_0')), \quad ((u_1, u_1'))\rangle \quad \langle((u_0, u_1')), ((u_1, u_0'))\rangle$
P_3	$M\overline{O}$	$(SR)\overline{L}$	(S,R)	$\langle((u_0, u_0'), (u_1, u_1')), ((u_1, u_1'), (u_0, u_0'))\rangle \langle((u_0, u_1'), (u_1, u_0')), ((u_1, u_0'), (u_0, u_1'))\rangle$
P_n'	$M\overline{O}[M\overline{L}]/$ $M\overline{O} - \lvert M\rvert$	Receiver notions	analogous	

The proofs in Table A.7 are done analogously to Lemma 2. This time, analogously proved relations are added in angle brackets.

Lemma 3. *P_B. For $X_1, X_2 \in \{S\overline{O}, S\overline{O} - |U|,\ S\overline{O} - H,\ S\overline{O} - P,\ SF\overline{L},\ SF\overline{L} - H,\ SF\overline{L} - P,\ SM\overline{L},\ SM\overline{L} - P\}$: If not $X_1 \implies X_2$ in our hierarchy, $(c, \epsilon, \delta) - X_1 \not\Longrightarrow (c^*, \epsilon^*, \delta^*) - X_2$ for any $\epsilon^* \geq 0, \delta^* < 1, c^* \geq 1$ and for any $\epsilon \geq 0, \delta < 1, c \geq 1$.*

Proof. If not $X_1 \implies X_2$ in our hierarchy, then in X_1 the adversary is more restricted, i.e. it exists a property *Prop* $\in \{U, |U|, Q, H, P\}$, which has to be equal in both scenarios for X_1, but neither has to be equal nor is implied to be equal for X_2 (See Figure A.2 to see which property can be used for which choice of X_1 and X_2).

We now assume a protocol Π, that achieves $(c, \epsilon, \delta) - X_1$. Let Π' be the protocol that additionally

Table A.7: Counter example construction idea with $X_1' = C\overline{O}$

P	X_1	X_2	I	attack
P_{10}-P_{15}	$(2S)\overline{O}$ $\langle(SM)\overline{O}$, $(SM)\overline{L}\rangle$	$(SR)\overline{O}$ $\langle(SR)\overline{L}\rangle$	1. (S, R)	$\langle((u_0, u_0')), ((u_1, u_1'))\rangle$, $\langle((u_0, u_1')), ((u_1, u_0'))\rangle$
P_{21}, P_{22}	$(2S)\overline{O}$	$(SM)\overline{O}$ $\langle(SM)\overline{L}\rangle$	1.$(S, \|m\|)$	$\langle((u_0, m_0)), ((u_1, m_1))\rangle$, $\langle((u_0, m_1)), ((u_1, m_0))\rangle$
$P_{n'}$	Receiver notions	analogous		

$$\begin{tikzcd}
S\overline{O} \arrow[rd, "|U|"] & & & \\
 & S\overline{O}-|U| \arrow[r, "H"] \arrow[d, "U"] & S\overline{O}-H \arrow[r, "P"] \arrow[d, "U"] & S\overline{O}-P \arrow[d, "U"] \\
 & SF\overline{L} \arrow[r, "H"] \arrow[rd, "Q"] & SF\overline{L}-H \arrow[r, "P"] \arrow[d, "Q"] & SF\overline{L}-P \arrow[d, "Q"] \\
 & & SM\overline{L} \arrow[r, "P"] & SM\overline{L}-P
\end{tikzcd}$$

Figure A.2: The properties at the implication arrows are restricted for the weaker notion (and all notions those notion implies), but not for the stronger notion.

publishes *Prop*. Π' still achieves $(c, \epsilon, \delta) - X_1$, since in all attacks on X_1 not more information than in Π are given to the adversary (The adversary knows *Prop* already since it is equal for both scenarios). However, since *Prop* does not have to be equal in X_2, the adversary can pick the scenarios such that it can distinguish them in Π' based on *Prop* with certainty. Hence for an arbitrary $\epsilon^* \geq 0, \delta^* < 1, n^* = c^* \geq 1$, $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', X_2, c^*, 0)\rangle] = 1$ and $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', X_2, c^*, 1)\rangle] = 0$ and thus $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', X_2, c^*, 0)\rangle] > e^{\epsilon^*} \cdot \Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', X_2, c^*, 1)\rangle] + \delta^*$ (since $1 > e^{\epsilon^*} \cdot 0 + \delta^*$). □

Lemma 4. *P_A. For $X_2 \in \{R\overline{O}, R\overline{O} - |U'|, R\overline{O} - H', R\overline{O} - P', RF\overline{L}, RF\overline{L} - H', RF\overline{L} - P', RM\overline{L}, RM\overline{L} - P'\}$, with $X_1 \not\Rightarrow X_2$: $(c, \epsilon, \delta) - S\overline{O}\{X_1\} \not\Rightarrow (c^*, \epsilon^*, \delta^*) - X_2$ for any $\epsilon^* \geq 0, \delta^* < 1, c^* \geq 1$ and for any $\epsilon \geq 0, \delta < 1, c \geq 1$.*

Proof. If $X_1 \not\Rightarrow X_2$, then it exists a property $Prop \in \{U', |U'|, Q', H', P'\}$, which has to be equal in both scenarios for X_1, but neither has to equal nor is implied to be equal in X_2 (see Proof to Lemma 3 for details).

We now assume a protocol Π, that achieves $(c, \epsilon, \delta) - S\overline{O}\{X_1\}$. Let Π' be the protocol that additionally outputs *Prop*. Since the properties of X_1 are also checked in attacks for $(c, \epsilon, \delta) - S\overline{O}\{X_1\}$ every valid attack on $(c, \epsilon, \delta) - S\overline{O}\{X_1\}$ will result in the same version of *Prop* for both scenarios. Hence, Π' does not output new information to the adversary (compared to Π) and still achieves $(c, \epsilon, \delta) - S\overline{O}\{X_1\}$.

However, since *Prop* does not have to be equal in X_2, the adversary can pick the scenario such that it can distinguish them with certainty in Π' based on *Prop*. Hence for an arbitrary $\epsilon^* \geq 0, \delta^* < 1, n^* = c^* \geq 1$, $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', X_2, c^*, 0)\rangle] = 1$ and $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', X_2, c^*, 1)\rangle] = 0$ and thus $\Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', X_2, c^*, 0)\rangle] > e^{\epsilon^*} \cdot \Pr[0 = \langle \mathcal{A} \mid Ch(\Pi', X_2, c^*, 1)\rangle] + \delta^*$ (since $1 > e^{\epsilon^*} \cdot 0 + \delta^*$). □

Lemma 5. *(Proofs P_{23}, P_{23}', P_{16}) $(SM)\overline{O} \not\Rightarrow (SM)\overline{L}$ [$(RM)\overline{O} \not\Rightarrow (RM)\overline{L}, (SR)\overline{O} \not\Rightarrow (SR)\overline{L}$ similarly]*

Proof. Protocol Construction: Given a protocol Π that achieves $(SM)\overline{O}$ and does not allow to recognize duplicated sender-message pairs, we construct protocol Π'. Let Π' run Π and additionally output a bit for every batch that contains only two communications. For some fixed senders $u_0 \neq u_1$ and messages $m_0 \neq m_1$ the protocol will additionally output a bit according to Table A.8. For any other batch with two communications it will pick the output bit randomly.

Table A.8: Additional output of Π'

(u_0, m_0)	(u_1, m_0)	(u_0, m_0)	(u_1, m_0)
(u_1, m_1)	(u_0, m_1)	(u_0, m_0)	(u_1, m_0)
(u_1, m_1)	(u_0, m_1)	(u_1, m_1)	(u_0, m_1)
(u_0, m_0)	(u_1, m_0)	(u_1, m_1)	(u_0, m_1)
output 0	output 1	output 1	output 0

Protocol Π' outputs the challenge bit b for $(SM)\overline{L}$ and hence does not achieve $(SM)\overline{L}$. However, it achieves $(SM)\overline{O}$. The strategy of entering the same challenge row twice does not work because the advantage gained in the cases of $(SM)\overline{L}$ (i.e. $a_1 \neq a_2$) is annihilated in the cases of duplicated communications (i.e. $a_1 = a_2$). Using one equal communication in both scenarios and a challenge row leads to another compensating distribution: The probability for a correct output is 0.25, for a wrong output 0.25 and for a random output 0.5. Another attack strategy is not possible since the additional output is only given for batches of size 2. □

Theorem 12. *For all X:*

1. $(c, \epsilon, \delta) - X \Rightarrow (c, \epsilon, \delta) - X_{CR_{c'}}$.
2. $(c, \epsilon, \delta) - X \Leftarrow (c, \epsilon, \delta) - X_{CR_{c'}}$, *if the number of challenges is not restricted.*

Proof. 1. Trivial: Given an attack valid on X restricted regarding the challenge rows per challenge. This attack is also valid against X without challenge row restriction.

2. We need to construct a new attack. *Attack construction:* Given an attack $\mathcal{A}_2$, we construct attack $\mathcal{A}_1$. Let $\bar{n}$ be the number of previous challenges used in $\mathcal{A}_1$ so far. For every batch query bq with n'' challenge rows replace the challenge tag of the 1st $, \dots, c'$st challenge row with $\bar{n}+1$; continue with the next c' challenge rows and the increased challenge tag until no challenge rows are left. Use all other queries as $\mathcal{A}_2$ does, give the answers to $\mathcal{A}_2$ and output whatever $\mathcal{A}_2$ outputs.

Given attack $\mathcal{A}_2$ on X. We construct an attack $\mathcal{A}_1$ with the same success against X_{CR_1} by using the above attack construction. We show that $\mathcal{A}_1$ is valid against X_{CR_1}: The attack construction assures, that at most c' challenge rows are used in every challenge ($CR_{c'}$). All other aspects of X are fullfilled in $\mathcal{A}_2$, too. Since $\mathcal{A}_1$ perfectly simulates the given attack $\mathcal{A}_2$, it has the same success. □

Remark. If a protocol does not achieve X_2 for c, then it does not achieve X_2 for any $c' > c$ (because every attack with only c CR is also valid if more than c CRs are allowed.) Further, notice that for some notions a minimum of two CRs is required (e.g. $(2S)\overline{O}$).

Additionally, for corruption options: Since the proposed attacks do not use any corruption, they are valid for any corruption option. Analogously, since the proposed attacks do not use different sessions in both scenarios, they are valid for any session option. □

A.6. On the Choice of Notions

The space of possible combinations of properties, and hence of conceivable privacy notions, is naturally large. Due to this, we verify our selection of privacy goals by finding exemple use cases. Additionally, we verify that our privacy notions include those of previous publications that suggest frameworks based on indistinguishability games, and provide a complete mapping in Section 3.6.1. We further demonstrate the choice and the applicability of our definition by analyzing the privacy goals of Loopix [128], an ACN that was recently published in Section 3.6.2.

Example Use Cases for the Notions

We illustrate our notions by continuing the example of an activist group trying to communicate in a repressive regime, although our notions are generally applicable.

Impartial Privacy Notions

These notions treat senders and receivers equally.

Message Observability. The content of messages can be learned in notions of this group, as messages are not considered confidential. Because the real world adversary can learn the content, we must prevent her from winning the game trivially by choosing different content. Hence, such notions use the property that the scenarios are identical except for the senders and receivers (E_{SR}) to ensure that the messages are equal in both scenarios.

EXAMPLE: *An activist of the group is already well-known and communication with that person leads to persecution of Alice.*

Alice needs a protocol that hides whether a certain sender and receiver communicate with each other; cf. Section 3.3.2 motivation of the complex property M_{SR}. The resulting notion is *Sender-Receiver Pair Unlinkability ($(SR)\overline{L}$).*

EXAMPLE (CONT.): *Only few people participate in the protocol. Then, just using the protocol to receive (send) something, when the well known activist is acting as sender (receiver) threatens persecution.*

Alice needs a protocol that hides whether a certain sender and receiver actively participate at the same time or not; cf. Section 3.3.2 motivation of the complex property R_{SR}. The resulting notion is *Sender-Receiver Unobservability ($(SR)\overline{O}$).*

Sender-Receiver Linkability (Message Confidentiality). Senders and receivers can be learned in notions of this group, because they are not considered private. Hence, such notions include the property that the scenarios are identical, except for the messages (E_M) to ensure that the sender-receiver pairs are equal in both scenarios.

EXAMPLE: *Alice wants to announce her next demonstration. (1) Alice does not want the regime to learn the content of her message and block this event. (2) Further, she is afraid that the length of her messages could put her under suspicion, e.g. because activists tend to send messages of a characteristic length.*

In (1) Alice needs a protocol that hides the content of the messages. However, the adversary is allowed to learn all other attributes, in particular the length of the message. Modeling this situation, the scenarios may differ solely in the message content; all other attributes must be identical in both scenarios, as they may not help the adversary distinguish between them. Beyond the above-described E_M, we must thus also request that the length of the messages $|M|$ is identical in both scenarios. The resulting notion is *Message Unobservability leaking Message Length ($M\overline{O} - |M|$)*[8].

In the second case (2), the protocol is required to hide the length of the message. The length of the messages thus may differ in the two scenarios, as the protocol will need to hide this attribute. Hence, we remove the restriction that the message length $|M|$ has to be equal in both scenarios from the above notion and end up with *Message Unobservability $M\overline{O}$.*

Both-Side Message Unlinkability. Notions of this group are broken if the sender-message or receiver-message relation is revealed.

EXAMPLE: *The activists know that their sending and receiving frequencies are similar to regime*

[8]We stick to our naming scheme here, although we would commonly call this confidentiality.

supporters' and that using an ACN is in general not forbidden, but nothing else. Even if the content and length of the message ($M\overline{O}$) and the sender-receiver relationship ($(SR)\overline{L}$) is hidden, the regime might be able to distinguish uncritical from critical communications, e.g. whether two activists communicate "Today" or innocent users an innocent message. In this case, the regime might learn that currently many critical communications take place and improves its measures against the activists.

In this case, the activists want a protocol that hides the communications, i.e. relations of sender, message and receiver. However, as using the protocol is not forbidden and their sending frequencies are ordinary, the adversary can learn which users are active senders or receivers and how often they sent and receive. Modeling this, the users need to have the same sending and receiving frequencies in both scenarios Q, Q', since it can be learned. However, everything else needs to be protected and hence, can be chosen by the adversary. This corresponds to the notion *Message Unobservability with Message Unlinkability ($M\overline{O}[M\overline{L}]$).*

Both-Side Unobservability. Even the activity of a certain sender or receiver is hidden in notions of this group.

Example (cont.): *It is a risk for the activists, if the regime can distinguish between two leading activists exchanging the message "today" and two loyal regime supporters exchanging the message "tomorrow".*

In this case, Alice wants to disclose nothing about senders, receivers, messages or their combination. However, the adversary can learn the total number of communications happening in the ACN. Modeling this, we need to assure that for every communication in the first scenario, there exists one in the second. We achieve this by prohibiting the use of the empty communication with property $\not\Diamond$. This results in the notion *Unobservability ($\overline{O}$).*

Example: *The regime knows that a demonstration is close, if the total number of communications transmitted over this protocol increases. It then prepares to block the upcoming event.*

To circumvent this, Alice needs a protocol that additionally hides the total number of communications. Modeling this, we need to allow the adversary to pick any two scenarios. Particularly, use of the empty communication $\Diamond$ is allowed. This is represented in the property that nothing needs to be equal in the two scenarios, $\aleph$, and results in the notion *Communication Unobservability ($C\overline{O}$).* Note that this is the only notion where the existence of a communication is hidden. All other notions include $\not\Diamond$ and hence do not allow for the use of the empty communication.

Sender (and Receiver) Privacy Notions

These notions allow a greater freedom in picking the senders (or receivers: similar notions are defined for receivers.).

Receiver-Message Linkability. The receiver-message relation can be disclosed in notions of this group. Hence, such notions include the property that the scenarios are identical except for the senders (E_S) to ensure the receiver-message relations are equal in both scenarios.

In *Sender-Message Unlinkability ($SM\overline{L}$)* the total number of communications and how often each user sends can be additionally learned. However, who sends which message is hidden. In *Sender-Frequency Unlinkability ($SF\overline{L}$)* the set of users and the total number of communications can be additionally disclosed. However, how often a certain user sends is hidden, since it can vary between the two scenarios. In *Sender Unobservability ($S\overline{O}$)*, the total number of communications can additionally be disclosed. However, especially the set of active senders U_b is hidden.

If a notion further includes the following abbreviations, the following information can be disclosed as well:

- *with User Number Leak* ($-|U|$): the number of senders that send something in the scenario

- *with Histogram Leak* ($-H$): the histogram of how many senders send how often
- *with Pseudonym Leak* ($-P$): which messages are sent from the same user

Example: *Alice is only persecuted when the regime can link a message with compromising content to her* – she needs a protocol that at least provides $SM\overline{L} - P$. *However, since such a protocol does not hide the message content, the combination of all the messages she sent might lead to her identification.* Opting for a protocol that additionally hides the message combination (P), i.e. provides $SM\overline{L}$, can protect her from this threat.
Further, assuming most users send compromising content, and Alice's message volume is high, the regime might easily suspect her to be the origin of some compromising messages even if she is careful that the combination of her messages does not reidentify her – she needs a protocol that does not disclose her sending frequencies (Q) although the combination of her messages (P) might be learned, i.e. achieving $SF\overline{L} - P$. *However, Alice might fear disclosing the combination of her messages* - then she needs a protocol achieving at least $SF\overline{L} - H$, which hides the frequencies (Q) and the message combination (P), but discloses the sending histogram, i.e. how many people sent how many messages (H). *However, if multiple activist groups use the ACN actively at different time periods, disclosing the sending histogram H might identify how many activist groups exist and to which events they respond by more active communication* – to prevent this she needs a protocol that hides the frequencies Q and the histogram H, i.e. provides $SF\overline{L}$.
Further, not only sending a certain content, but also being an active sender (i.e. being in U) is prosecuted she might want to pick a protocol with at least $S\overline{O} - P$. Again if she is afraid that leaking P or H together with the expected external knowledge of the regime would lead to her identification, she picks the corresponding stronger notion. *If the regime knows that senders in the ACN are activists and learns that the number of active senders is high, it blocks the ACN.* In this case at least $S\overline{O}$ should be picked to hide the number of senders ($|U|$).

Example: *For the next protest, Alice sends two messages: (1) a location, and (2) a time. If the regime learns that both messages are from the same sender, they will block the place at this time even if they do not know who sent the messages.* Alice then needs a protocol that hides whether two communications have the same sender or not. We already explained how to model this with complex property T_S in Section 3.3.2. The resulting notion is Twice Sender Unobservability ($(2S)\overline{O}$).

Receiver Observability. In notions of this group the receiver of each communication can be learned. Hence, such notions include the property that the scenarios are equal except for the senders and messages (E_{SM}) to ensure that they are equal in both scenarios.

Example: *Consider not only sending real messages is persecuted, but also the message content or any combination of senders and message contents is exploited by the regime. If the regime e.g. can distinguish activist Alice sending "today" from regime supporter Charlie sending "see u", it might have learned an information the activists would rather keep from the regime. Further, either (1) the activists know that many messages of a certain length are sent or (2) they are not sure that many messages of a certain length are sent.*

In case (1), Alice needs a ACN, that hides the sender activity, the message content and their combination. However, the adversary can especially learn the message length. Modeling this, beyond the above described E_{SM}, the message lengths have to be equal $|M|$. This results in the notion *Sender Unobservability with Message Unobservability leaking Message Length* ($S\overline{O}[M\overline{O} - |M|]$). Note that in $S\overline{O}[M\overline{O} - |M|]$ the properties of $M\overline{O} - |M|$ are included and further the senders are allowed to differ in the two scenarios. The second case (2) requires a protocol that additionally hides the message length. Hence, in modeling it we remove the property that the message lengths are equal $|M|$ from the above notion. This results in *Sender Unobservability with Message Unobservability* ($S\overline{O}[M\overline{O}]$).

Example: *Alice's demonstration is only at risk if the regime can link a message with a certain content to her as a sender with a non negligible probability.* Then at least *Sender-Message Pair Unlinkability ($(SM)\overline{L}$)*, which is defined analogous to $(SR)\overline{L}$ is needed.

Example (cont.): *However, $(SM)\overline{L}$ only allows Alice to claim that not she, but Charlie sent a critical message m_a and the regime cannot know or guess better. Now assume that Dave is also* *communicating, then the regime might be able to distinguish Alice sending m_a, Charlie m_c and Dave m_d from Alice sending m_d, Charlie m_a and Dave m_c. In this case, it might not even matter that Alice can claim that Charlie possibly sent her message. The fact that when comparing all three communications that possibly happened, Alice is more likely to have sent the critical message m_a means a risk for her.*

To circumvent this problem Alice needs a protocol that not only hides the difference between single pairs of users, but any number of users. Modeling this, instead of the complex property M_{SM}, we need to restrict that the active senders' sending frequencies are equal, i.e. $SM\overline{L}$.

Example: *In another situation our activists already are prosecuted for being a sender while a message with critical content is sent.*

In this case at least *Sender-Message Pair Unobservability ($(SM)\overline{O}$)*, which is defined analogous to $(SR)\overline{O}$ is needed.

Analogous notions are defined for receivers.

Sender Privacy Notions: Both-Side Message Unlinkability. As explained with the example before in the case that Alice does not want any information about senders, receivers and messages or their combination to leak, she would use $\overline{O}$. However, the privacy in this example can be tuned down, if she assumes that the regime does not have certain external knowledge or that the users are accordingly careful. As explained for the Sender Notions with Receiver-Message Linkability before, in this case we might decide to allow $U', |U'|, Q', H', P'$ to leak.

If a notion $X \in \{R\overline{O}, R\overline{O}-|U'|, R\overline{O}-H', R\overline{O}-P', RF\overline{L}, RF\overline{L}-H', RF\overline{L}-P', RM\overline{L}, RM\overline{L}-P'\}$ is extended to *Sender Unobservability by X ($S\overline{O}\{X\}$)*, the leaking of the sender-message relation is removed. This is done by removing E_R. Since the attacker now has a greater degree of freedom in choosing the senders and is (if at all) only restricted in how she chooses the receivers and messages, this is a special strong kind of Sender Unobservability. Analogous notions are defined for receivers.[9]

A.7. How to Use

The framework described above offers the opportunity to thoroughly analyze ACNs. To perform such an analysis, we advice a top-down approach as follows.

1. In case the ACN under analysis can be instantiated to protect against different adversaries, fix those parameters.

2. Extract capabilities of the adversary and general protocol properties from the ACN description: Specify the allowed *user corruption*. Is it none, static, adaptive? See Table 3.4. Are *sessions* (channels) constructed that link messages from the same sender? See Section A.1.1. Extract all other capabilities to include them in the protocol model.

3. Simplify the ACN protocol in a protocol model: Generate a simplified protocol (*ideal functionality*) without cryptography by assuming secure communication. Show indistinguishability between this ideal functionality and the real-world protocol using a *simulation based proof*. Previous work [9] can guide the modeling step. See Section 3.6.1 (UC-realizability) for how the result of the simplified protocol can be transferred to the real-world protocol.

4. Extract properties based upon the input to the adversary from the ideal functionality: Start with *simple properties*, see Table 3.1. What does the adversary learn from the protocol execution? Continue with *complex properties*. See Section 3.3.2.

5. After mapping all properties from the protocol and adversary model, a privacy notions must be selected. Either the description of the ACN already specifies (in-)formally which privacy

[9]Note that $S\overline{O}\{R\overline{O}\} = R\overline{O}\{S\overline{O}\} = \overline{O}$.

goal should be achieved, or the ACN under analysis should be shown to achieve a certain notion. See Table 3.2 for an overview of our defined notions.

6. As it is easier to show that a certain notion is not fulfilled compared to show that it is fulfilled, we propose to start with the strongest notions extracted this way. A notion is not fulfilled if the functionality (and thus the protocol) leaks a property to the adversary that he is not allowed to learn for the given notion. If it is not obvious that a notion is not fulfilled, check if the notion can be proven for the protocol model. The related work of Gelernter and Herzberg [72] and Backes et al. [11] serve as examples for such proofs.

If the proof cannot be constructed or $\delta = 1$, a weaker notion can be selected for analysis. In case that the proof goes through and yields $\epsilon = 0$ and a negligible δ, the protocol was shown to achieve the selected notion as per Definition 2. If $\epsilon > 0$ or a non negligible $\delta < 1$, the protocol achieves the selected notion as per Definition 42.

If the protocol supports different adversaries, the steps described above can be repeated. This typically leads to adjusting the ideal functionality or adding different adversary classes (see Section 3.4) and thus fulfilling different properties of our framework. Analysis results under a variation of ACN parameters may achieve different notions in our hierarchy (Figure 3.3 and Figure 3.4). Based on our established relations between notions, analysis results can be compared for various parameters or parameter ranges, as well as against results of other ACNs.

A.8. Primitives' Details

A.8.1 Pseudocode

For our pseudocode representation, we extend the scenario model to not only capture communications from senders to receivers, but also from an to intermediate nodes (proxies) $P, P_{u_1}, P_{u_2}, \dots$. Primitives know the difference between intermediate nodes and users and can act upon it. We detail a representative selection of our primitives in Algorithm 3 - 11. In general, we use $m^i_{null} := \{0\}^i$ to represent the specific message consisting of exactly i zero bits ($|m^i_{null}| = i$), $m_{null} := m^x_{null}$ for some default length x, e.g. the one that is being padded to, and $aux_{null} := 0$ as the information that just consist of a zero bit, μ for the maximum number of communications, S as the set of all senders and U as the set of active senders in the input batch ($U = \{u \mid (u, \_, \_, \_) \in \underline{r}\}$).

Algorithm 3: `EncToMix`

```
Upon query $\underline{r}$
    for $comm = (u, u', m, aux) \in \underline{r}$ do
        if $u' == P \lor u' == P_x$ for any $x$ then
            $i = |m|$
            $comm' = (u, u', m^i_{null}, aux)$
            $replace(comm, comm')$
    output $\underline{r}$
```

Algorithm 4: `S2S`

```
Upon query $\underline{r}$
    $comm = (Source, Sink, m_{null}, aux_{null})$
    while $length(\underline{r}) < \mu$ do
        $append(\underline{r}, comm)$
    output $\underline{r}$
```

Algorithm 5: `AllSend`

```
Upon query $\underline{r}$
    $\hat{S} = S \setminus U$
    for $u_j \in \hat{S}$ do
        $comm = (u_j, Sink, m_{null}, aux_{null})$
        $append(\underline{r}, comm)$
    output $\underline{r}$
```

Algorithm 6: `k − rdmS`

```
Upon query $\underline{r}$
    $\hat{S} = S \setminus U$
    $k = length(\underline{r}) - |U|$
    $j = 1$
    while $j < k \wedge \hat{S} \neq \emptyset$ do
        $u_j \xleftarrow{r} \hat{S}$
        $\hat{S} = \hat{S} \setminus \{u_j\}$
        $U = U \cup\ u_j$
        $j + +$
    for $u_j \in U$ do
        $comm = (u_j, Sink, m_{null}, aux_{null})$
        $append(\underline{r}, comm)$
    output $\underline{r}$
```

Algorithm 7: $\mathtt{D2S}_{\mathtt{System}}$

```
Upon query $\underline{r}$
    $Paddings = \{(u : \mu) | u \in S\}$
    for $(u_i, \_, \_, \_) \in \underline{r}$ do
        $Paddings[u_i] - = 1$
    $\underline{r}_{dummy} = []$
    for $(u_i : pad_i) \in Paddings$ do
        for $j \in\ [1, .., pad_i]$ do
            $comm = (u_i, Sink, m_{null}, aux_{null})$
            $insertRandom(\underline{r}_{dummy}, comm)$
    $\underline{r} + = \underline{r}_{dummy}$
    output $\underline{r}$
```

Algorithm 8: `D2A`

```
Upon query $\underline{r}$
    $\underline{r}_{Result} = []$
    for $comm = (u_i, u'_i, m, \_) \in \underline{r}$ do
        for $u' \in R$ do
            if $u' == u'_i$ then
                $append(\underline{r}_{Result}, comm)$
            else
                $j = |m|$
                $comm' = (u_i, u', m^j_{null}, aux_{null})$
                $append(\underline{r}_{Result}, comm')$
    output $\underline{r}_{Result}$
```

Algorithm 9: `Delay`

```
Upon query $\underline{r}$
    $\underline{r}_{Result} = []$
    for $comm_i \in \underline{r}$ do
        $insertRandom(\underline{r}_{Result}, comm_i)$
    output $\underline{r}_{Result}$
```

Algorithm 10: `OutboxMix`

```
Upon query $\underline{r}$
    $\underline{r}_{Forward} = []$
    for $comm_i = (u_i, u'_i, m_i, aux_i) \in \underline{r}$ do
        if $u_i \in Proxy$ then
            $insertRandom(\underline{r}_{Forward}, comm_i)$
            $remove(comm_i, \underline{r})$
        else
            if $u'_i \in Proxy$ then
                skip
            else
                $comm'_i = (u_i, P_{u_i}, m_i, aux_i)$
                $replace(comm_i, comm'_i)$
                $comm''_i = (P_{u_i}, u'_i, m_i, aux_i)$
                $insertRandom(\underline{r}_{Forward}, comm''_i)$
    $\underline{r} = \underline{r} + \underline{r}_{Forward}$
    output $\underline{r}$
```

Algorithm 11: `SameAddr`

```
Upon query $\underline{r}$
    if $P_x \in \underline{r}$ for any $x$ then
        replace $P_x$ with $P$ for all $x$
    else
        for $comm = (u_i, u'_i, m_i, aux_i) \in \underline{r}$ do
            $comm' = (u_i, P, m_i, aux_i)$
            $comm'' = (P, u'_i, m_i, aux_i)$
            $replace(comm, comm')$
            $append(\underline{r}, comm'')$
    output $\underline{r}$
```

A.8.2 Protection

Idea

$X_{weak} \overset{\texttt{prim}}{\Longrightarrow} X_{strong}$: Assume a successful attack $\mathcal{A}_{X_{strong}}$ on X_{strong} for `prim`(ACN). Show that there also is a successful attack $\mathcal{A}_{X_{weak}}$ on X_{weak} for ACN (i.e. the condition of the implication is invalid).

Dummy Primitives

`pad`. $M\overline{O} - |m| \overset{\texttt{pad}}{\Longrightarrow} M\overline{O}$: Given an attack $\mathcal{A}_{M\overline{O}}$ on `pad`(ACN). Attack $\mathcal{A}_{M\overline{O}-|m|}$ on ACN uses the same batches as $\mathcal{A}_{M\overline{O}}$ except that it applies `pad` before submitting them. Thus the messages in the submitted batches all have the same length and $\mathcal{A}_{M\overline{O}-|m|}$ is valid for $M\overline{O} - |m|$. Otherwise $\mathcal{A}_{M\overline{O}-|m|}$ uses the same algorithm as $\mathcal{A}_{M\overline{O}}$. Notice that the scenarios handed to the protocol model and hence the probability distribution of adversarial observations is the same for $\mathcal{A}_{M\overline{O}-|m|}$

on ACN and $\mathcal{A}_{\overline{MO}}$ on $\texttt{pad}(ACN)$.

$[\overline{SO}[\overline{MO} - |m|] \xRightarrow{\texttt{pad}} \overline{SO}[\overline{MO}]$, $\overline{RO}[\overline{MO} - |m|] \xRightarrow{\texttt{pad}} \overline{RO}[\overline{MO}]$ similar]

$\texttt{S2S}$. $\overline{O} \xRightarrow{\texttt{S2S}} \overline{CO}$: Similar as $\texttt{pad}$: $\texttt{S2S}$ ensures that no empty communications take place. Hence after applying it the inputs chosen by $\mathcal{A}_{\overline{CO}}$ are valid for $\overline{O}$.

$\texttt{AllSend}$. $\overline{SFL} \xRightarrow{\texttt{AllSend}} \overline{SO} - |U|$: Similar as $\texttt{pad}$: $\texttt{AllSend}$ ensures that any user sends at least once. Hence after applying it the inputs chosen by $\mathcal{A}_{\overline{SO}-|U|}$ are valid for $\overline{SFL}$, because $U = \mathcal{U}$.

$[\overline{SFL} - H \xRightarrow{\texttt{AllSend}} \overline{SO} - H$, $\overline{SFL} - P \xRightarrow{\texttt{AllSend}} \overline{SO} - P$, $\overline{RO}\{\overline{SFL}\} \xRightarrow{\texttt{AllSend}} \overline{RO}\{\overline{SO} - |U|\}$, $\overline{RO}\{\overline{SFL} - H\} \xRightarrow{\texttt{AllSend}} \overline{RO}\{\overline{SO} - H\}$, $\overline{RO}\{\overline{SFL} - P\} \xRightarrow{\texttt{AllSend}} \overline{RO}\{\overline{SO} - P\}$ and for the receiver primitive $\texttt{AllRec}$ similarly.]

$\texttt{D2S}_{\texttt{System}}$. $\overline{SML} \xRightarrow{\texttt{D2S}_{\texttt{System}}} \overline{SO}$. Similar as $\texttt{pad}$: $\texttt{D2S}_{\texttt{System}}$ ensures that any user sends the system (delivery) maximum of messages by padding with messages to the sink. After applying it the inputs chosen by $\mathcal{A}_{\overline{SO}}$ are valid for $\overline{SML}$, as the first system maximum real messages are delivered and all users send (dummy and real) with the same frequency. Notice that this especially keeps the messages and receivers of each communication equal to the corresponding communication in the other scenario [10].

$[\overline{MO}[\overline{ML}] \xRightarrow{\texttt{D2S}_{\texttt{System}}} \overline{SO}\{\overline{RML}\}$, $\texttt{DfS}_{\texttt{System}}$ similarly.]

$\texttt{D2S}_{\texttt{User}}$. $\overline{RO}\{\overline{SML}\} \xRightarrow{\texttt{D2SP}_{\texttt{User}}} \overline{CO}$. Similar as $\texttt{D2S}_{\texttt{System}}$: $\texttt{D2S}_{\texttt{User}}$ pads each users sending events to the fixed user maximum and thereby ensures the same sending frequencies and the same total number of (dummy and real) messages. Thus after applying it the inputs chosen by $\mathcal{A}_{\overline{CO}}$ are valid for $\overline{RO}\{\overline{SML}\}$, as $\overline{RO}\{\overline{SML}\}$ does not require anything else.

$\texttt{D2SP}_{\texttt{Classes}}$. $\overline{SFL} - H \xRightarrow{\texttt{D2SP}_{\texttt{Classes}}} \overline{SFL}$. Similar as $\texttt{pad}$: $\texttt{D2SP}_{\texttt{Classes}}$ ensures that users either send no message or adhere to their class[11] and hence after applying it the inputs chosen by $\mathcal{A}_{\overline{SFL}}$ are valid for $\overline{SFL} - H$.

$[\overline{SO} - H \xRightarrow{\texttt{D2SP}_{\texttt{Classes}}} \overline{SO} - |U|$, $\overline{RO}\{\overline{SFL} - H\} \xRightarrow{\texttt{D2SP}_{\texttt{Classes}}} \overline{RO}\{\overline{SFL}\}$, $\overline{RO}\{\overline{SO} - H\} \xRightarrow{\texttt{D2SP}_{\texttt{Classes}}} \overline{RO}\{\overline{SO} - |U|\}$, $\texttt{DfSP}_{\texttt{Classes}}$ similarly]

$\texttt{D2SP}_{\texttt{Traffic}}$. $\overline{SML} \xRightarrow{\texttt{D2SP}_{\texttt{Traffic}}} \overline{SFL} - H$. Similar as $\texttt{pad}$: $\texttt{D2SP}_{\texttt{Traffic}}$ ensures that users either send no message or adhere to the maximum of messages per sender in the batch[12] and hence after applying it the users in U will sent the traffic maximum (that is the same in both batches due to

[10]Note that $\texttt{D2S}_{\texttt{User}}$ does not as there the delivered real messages and the frequency per receiver can differ between the scenarios after applying the primitive. Assume as first scenario the one where Alice wants to send all messages and the other where all users only want to send the user maximum of messages. After applying $\texttt{D2S}_{\texttt{User}}$ the first scenario will deliver the first user maximum messages from Alice to the intended receivers, but not the other messages. The second scenario will deliver all messages to the intended receivers.

[11]As the biggest class allows to send the system maximum of messages and only the first system maximum messages are considered for the ranking to classes, exactly these messages will be real messages in the adapted input and hence E_S holds for the adapted input too. Notice that $\texttt{D2SP}_{\texttt{Classes}}$ does not modify the set of active senders U (calculated over the whole batch not just the first system maximum of (real) communications.)

[12]In the extreme case this requires to pad to the system maximum for all, but always the first system maximum communications are delivered and hence E_S holds for the adapted input too. Notice that $\texttt{D2SP}_{\texttt{Traffic}}$ does not modify the set of active senders U.

H) and all others no (real or dummy) messages, fulfilling Q. Thus the inputs chosen by $\mathcal{A}_{S\overline{FL}-H}$ are after adaption by $\texttt{D2SP}_{\texttt{Traffic}}$ valid for $S\overline{ML}$.

[$S\overline{ML}-P \xrightarrow{\texttt{D2SP}_{\texttt{Traffic}}} S\overline{FL}-P$ (as P ensures the same maximum by implying H), $R\overline{O}\{S\overline{ML}\} \xrightarrow{\texttt{D2SP}_{\texttt{Traffic}}} R\overline{O}\{S\overline{FL}-H\}$, $R\overline{O}\{S\overline{ML}-P\} \xrightarrow{\texttt{D2SP}_{\texttt{Traffic}}} R\overline{O}\{S\overline{FL}-P\}$, $\texttt{DfSP}_{\texttt{Traffic}}$ similarly]

$\texttt{D2SP}_{\texttt{System}}$. $S\overline{ML} \xrightarrow{\texttt{D2SP}_{\texttt{System}}} S\overline{FL}$. Similar as $\texttt{pad}$: $\texttt{D2SP}_{\texttt{System}}$ ensures that users either send no message or adhere to the system maximum[13] and hence after applying it the inputs chosen by $\mathcal{A}_{S\overline{FL}}$ are valid for $S\overline{ML}$, as they fulfill Q.

$\texttt{D2SP}_{\texttt{User}}$. $R\overline{O}\{S\overline{ML}\} \xrightarrow{\texttt{D2SP}_{\texttt{User}}} R\overline{O}\{S\overline{FL}\}$. Similar as $\texttt{D2SP}_{\texttt{System}}$: $\texttt{D2SP}_{\texttt{User}}$ pads each active users' sending events to the fixed user maximum and thereby ensures the same sending frequencies under active users and no sending events for inactive users, fulfilling Q, and thereby ensures the same sending frequencies and the same total number of (dummy and real) messages. Thus after applying $\texttt{D2SP}_{\texttt{User}}$ the inputs chosen by $\mathcal{A}_{R\overline{O}\{S\overline{FL}\}}$ are valid for $R\overline{O}\{S\overline{ML}\}$, as $R\overline{O}\{S\overline{ML}\}$ does not require anything else.[14]

$\texttt{D2A}$. $M\overline{O} \xrightarrow{\texttt{D2A}} R\overline{O}[M\overline{O}]$. Similar as $\texttt{pad}$: $\texttt{D2A}$ ensures that all receivers receive the same number of times (in the same order) and hence after applying it the inputs chosen by $\mathcal{A}_{R\overline{O}[M\overline{O}]}$ are valid for $M\overline{O}$.

[$M\overline{O}-|m| \xrightarrow{\texttt{D2A}} R\overline{O}[M\overline{O}-|m|]$ and $\texttt{DfA}$ similarly]

$\texttt{k}-\texttt{rdmS}$. $S\overline{O}-|U| \xrightarrow{\texttt{k}-\texttt{rdmS}} S\overline{O}$. Similar as $\texttt{pad}$: $\texttt{k}-\texttt{rdmS}$ ensures that as many users send as real messages are sent, thus $|U|$ is fulfilled after applying the primitive and as only dummy messages to the sink are added E_S still holds. Hence, after applying it the inputs chosen by $\mathcal{A}_{S\overline{O}}$ are valid for $S\overline{O}-|U|$.

[$R\overline{O}\{S\overline{O}-|U|\} \xrightarrow{\texttt{k}-\texttt{rdmS}} \overline{O}$, $\texttt{k}-\texttt{rdmR}$ similarly]

$\texttt{E2EC}$. $S\overline{O} \xrightarrow{\texttt{E2EC}} S\overline{O}[M\overline{O}-|m|]$. Similar as $\texttt{pad}$: $\texttt{E2EC}$ replaces all messages with a receiver-specific null message and thus ensures that the same messages are used in both cases (because the same receivers are used). Hence, after applying it the inputs chosen by $\mathcal{A}_{S\overline{O}[M\overline{O}-|m|]}$ are valid for $S\overline{O}$.

Notice that without key-privacy of the encryption scheme, the replacement messages are receiver dependent.

$\texttt{E2EC(kp)}$. Null $\xrightarrow{\texttt{E2EC(kp)}} M\overline{O}-|m|$. Similar as $\texttt{E2EC}$, but now all messages are replaced with the same null message (of the same length as the message before, but receiver independent). This hides message content completely. Hence, after applying it the inputs chosen by $\mathcal{A}_{M\overline{O}-|m|}$ are valid for Null, i.e. identical.

[$S\overline{ML}-P \xrightarrow{\texttt{E2EC(kp)+pad}} S\overline{ML}$, $S\overline{FL}-P \xrightarrow{\texttt{E2EC(kp)+pad}} S\overline{FL}-H$, $S\overline{O}-P \xrightarrow{\texttt{E2EC(kp)+pad}} S\overline{O}-H$, $R\overline{ML}-P' \xrightarrow{\texttt{E2EC(kp)+pad}} R\overline{ML}$, $R\overline{FL}-P' \xrightarrow{\texttt{E2EC(kp)+pad}} R\overline{FL}-H'$, $R\overline{O}-P' \xrightarrow{\texttt{E2EC(kp)+pad}} R\overline{O}-H'$,

[13] As before exactly the first system maximum communications will be delivered in the adapted input and hence E_S holds for the adapted input too. Notice that $\texttt{D2SP}_{\texttt{System}}$ does not modify the set of active senders U (calculated over the whole batch not just the first system maximum of (real) communications.)

[14] Notice that this primitive however does not guarantee to keep E_S which $\texttt{D2SP}_{\texttt{System}}$ provides.

$R\overline{O} \xRightarrow{\texttt{E2EC(kp)}} R\overline{O}[M\overline{O} - |m|]$, $S\overline{O}\{RM\overline{L} - P\} \xRightarrow{\texttt{E2EC(kp)+pad}} S\overline{O}\{RM\overline{L}\}$, $S\overline{O}\{RF\overline{L} - P\} \xRightarrow{\texttt{E2EC(kp)+pad}} S\overline{O}\{RF\overline{L} - H\}$, $S\overline{O}\{R\overline{O} - P\} \xRightarrow{\texttt{E2EC(kp)+pad}} S\overline{O}\{R\overline{O} - H\}$, $R\overline{O}\{SM\overline{L} - P\} \xRightarrow{\texttt{E2EC(kp)+pad}} R\overline{O}\{SM\overline{L}\}$, $R\overline{O}\{SF\overline{L} - P\} \xRightarrow{\texttt{E2EC(kp)+pad}} R\overline{O}\{SF\overline{L} - H\}$, $R\overline{O}\{S\overline{O} - P\} \xRightarrow{\texttt{E2EC(kp)+pad}} R\overline{O}\{S\overline{O} - H\}$ similarly]

Note that `pad` ensures that messages are the same length before they are replaced with the null message. Note further, that key-privacy is required both for receiver notions (as otherwise the corresponding receiver can leak based on the observed null message) and for sender notions as otherwise P will not be equal after applying (consider e.g. A sending to R_1 and R_2 in the first and to R_1 and R_3 in the second (and B sending to the remaining one)).

`Delay`. $S\overline{O}[M\overline{O}] \xRightarrow{\texttt{Delay}} S\overline{O}\{RM\overline{L}\}$. Given an attack $\mathcal{A}_{S\overline{O}\{RM\overline{L}\}}$ on `Delay`(ACN). Attack $\mathcal{A}_{S\overline{O}[M\overline{O}]}$ on ACN uses the same batches as $\mathcal{A}_{S\overline{O}\{RM\overline{L}\}}$ except that it

1. picks a random permutation of the (first system maximum) communications in the first scenario,
2. sorts the communications of the second scenario to have the same receiver as in the first (note that due to Q' this has to work) and decides randomly whenever there still is a degree of freedom and
3. submits them to the challenger.

Thus the constructed batches are valid for $S\overline{O}[M\overline{O}]$ and use any permutation with the same probability as `Delay` would. Otherwise $\mathcal{A}_{S\overline{O}[M\overline{O}]}$ uses the same algorithm as $\mathcal{A}_{S\overline{O}\{RM\overline{L}\}}$. Notice that the scenarios handed to the protocol model and hence the probability distribution of adversarial observations is the same for $\mathcal{A}_{S\overline{O}[M\overline{O}]}$ on ACN and $\mathcal{A}_{S\overline{O}\{RM\overline{L}\}}$ on `Delay`(ACN).

[$R\overline{O}[M\overline{O}] \xRightarrow{\texttt{Delay}} R\overline{O}\{SM\overline{L}\}$, `Delay`(leak) similar]

`Delay+ SameAddr+ AnyMix`. $M\overline{O} \xRightarrow{\texttt{Delay+ SameAddr+ AnyMix}} M\overline{O}[M\overline{L}]$. Similar as `Delay`. Additionally, `SameAddr+ AnyMix` ensure that only the same sender-receiver pairs are used because Q and Q' are given and after applying first `AnyMix` and then `SameAddr` for every communication every sender sends once to the proxy and the proxy once to the receiver. Finally `Delay` removes any order dependency as above and the attack $\mathcal{A}_{M\overline{O}}$ can pick a random permutation for the first scenario and the corresponding sender-receiver combinations for the second (as the same pairs are used after applying `SameAddr+ AnyMix`).

`Delay+ EncToMix+ pad+ SameAddr+ AnyMix`. Null $\xRightarrow{\texttt{Delay+ EncToMix+ pad+ SameAddr+ AnyMix}} SM\overline{L}$.

Similar to `Delay+ SameAddr+ AnyMix`. Additionally using `EncToMix+ pad` ensures that the null message of the same length is used for all communications and hence the attack $\mathcal{A}_{\text{Null}}$ can pick a random permutation for the first scenario, but sort the communications of the second to be identical, as sender-receiver pairs are equal due to `SameAddr+ AnyMix` and all use the same message due to `EncToMix+ pad`.

[Null $\xRightarrow{\texttt{EncToMix+ pad+ SameAddr+ AnyMix}} RM\overline{L}$ similar, but we do not need `Delay` as the (real, not proxy) senders appear in the same order already and the proxy communications to the real receivers are in a random order due to `AnyMix` already.]

`SameAddr+ AnyMix`. Null $\xRightarrow{\texttt{SameAddr+ AnyMix}} (SR)\overline{L}$. Given attack $\mathcal{A}_{(SR)\overline{L}}$ on $(SR)\overline{L}$ of `SameAddr+ AnyMix`(ACN). In the attack for $(SR)\overline{L}$ each sender sends in the same order in both scenarios (for any instance) and the challenge row users even send the same message. We construct an attack $\mathcal{A}_{\text{Null}}$ in the following way: $\mathcal{A}_{\text{Null}}$ throws a fair random coin to decide on the instance.

Then it applies `SameAddr`+ `AnyMix`, which ensures the same communications *to* the proxy start in the same order from the real senders. The communications from the proxy to the real receivers are appended in random order. As for `Delay` before, $\mathcal{A}_{\text{Null}}$ decides on a random order for these communications in the first and use the same order in the second scenario (which uses each order with the same probability as the `AnyMix` primitive does). It hands these changed batches to the challenger and does anything else as $\mathcal{A}_{\text{Null}}$. Notice that the new input scenarios are identical and thus valid for Null and the attack produces the same inputs to ACN as $\mathcal{A}_{(SR)\overline{L}}$ and hence has the same success.

`InboxMix`+ `EncToMix`+ `pad`. Null $\xRightarrow{\texttt{InboxMix+ EncToMix+pad}}$ $(SM)\overline{L}$. Similar to `SameAddr`+ `AnyMix`. `pad`+ `EncToMix` ensures now that all messages for communications from the sender to any proxy are identical and `InboxMix` makes the sender-receiver pairs identical as the challenge rows already use the same receiver. Notice that due to the random order of communications outgoing of the proxy, the communications to the receiver can be ordered to be identical, even though the messages for these communications are not.

[Null $\xRightarrow{\texttt{OutboxMix+ EncFromMix+ pad}}$ $(RM)\overline{L}$ similar, but the random order in the communications from the sender to the proxy that allows us to sort real sender communications to be identical is given by the instances in $(RM)\overline{L}$, not the primitive. So the attack will not use both scenarios for $i = 0$ and $i = 1$ as before, but pick $i = 0, b = 0$ and $i = 1, b = 1$ with 50% and $i = 0, b = 1$ and $i = 1, b = 0$ with 50% (leading to the same probabilities that any of them is picked for the challenge). Leaky versions similar.]

Constructions

`Broadcast`. $Null \xRightarrow{\texttt{Broadcast}} R\overline{O}$. Similar to `D2A`: `Broadcast` ensures that all receivers receive the same message (in the same order) and hence after applying it the inputs chosen by $\mathcal{A}_{R\overline{O}}$ are valid for Null.

[Null $\xRightarrow{\texttt{reverseBroadcast}}$ $S\overline{O}$ similar]

Staged `Broadcast`. Null $\xRightarrow{\text{Staged }\texttt{Broadcast}}$ $(2R)\overline{O}$. Similar as `Broadcast`. Given an attack $\mathcal{A}_{(2R)\overline{O}}$ on $(2R)\overline{O}$ of Staged `Broadcast`(ACN). The attack on $(2R)\overline{O}$ already uses the same communications in the first stage (for any instance) and only differs in the receiver of the challenge communication in the second stage (for each instance). By applying Staged `Broadcast`, any communication of the second stage is sent to all receivers in the same order and hence the communications of the second stage are equal. The attack on Null only has to randomly pick one instance and apply `Broadcast` to the second stage to be valid for Null and have the same success as $\mathcal{A}_{(2R)\overline{O}}$.

Combinations

Nearly all of the dotted implications only add the missing primitives to reach the higher notion, e.g. $(SM)\overline{L} \xRightarrow{\texttt{Delay+ SameAddr}} SM\overline{L}$ adds the primitives from Null $\xRightarrow{\texttt{Delay+ EncToMix+ pad+ SameAddr+ AnyMix}}$ $SM\overline{L}$ that are missing in Null $\xRightarrow{\texttt{InboxMix+ EncToMix+pad}}$ $(SM)\overline{L}$.

Only for $(SM)\overline{L} \xRightarrow{\texttt{EncFromMix}} M\overline{O}$ (and the receiver equivalent) we need to further notice that replacing the messages with the null message before and after the proxy (`EncToMix`+`EncFromMix`) results in the same message replacement as replacing all messages with null messages (`E2EC(kp)`).

A.9. Dissecting Systems to Primitive Combinations

We relate four ACNs to our primitives: a basic Mixnet, DC-Net, Pung and Loopix. We decide on those because they employ different basic strategies (DC-Nets, Mixnets, PIR) and except for the Mixnet as first example, all employ some dummy traffic, ensuring interesting primitive combinations. For each of the ACNs, we summarize their design, explain the mapping of it to primitives and discuss the corresponding privacy notions.

We stress that our intention is an initial investigation of the decomposition into primitives and not to provide a security proof for the protocols.

A.9.1 Mixnet

We start with the mapping of a basic mixnet (adapted from [42] and part of [21, 128, 44]) as a first example.

Description. A basic mixnets protects the sender's privacy against the GPA. Senders use multiple layers of encryption and sent the (encrypted) fixed-size message over a sequence of mixes. At every mix one layer of encryption is removed. Packets are further delayed until enough are collected and the processed packets output in a random order. The packet only includes the receiver's address in the innermost layer of encryption. To hide the order of sending events per user, we assume that the senders wait for a random time before sending the message into the network.

Primitives. The fixed message length corresponds to `pad`. The mix uses `SameAddr`, as packets from all senders can use the same mix and the final address is only learned by the last mix. As the order of outgoing packets does not depend on the incoming packets it also unlinks in the timing dimension, corresponding to `InboxMix` (and `OutboxMix`[15]) and thus `AnyMix`. The layered encryption corresponds to `EncToMix` only, as the mixnet does not encrypt the message for the receiver. Finally, as senders wait for a random time before they input their message into the network, it further implements `Delay`.

Notions. On the sender side, basic mixnets as described above result in $SM\overline{L}$ that implies both $(SR)\overline{L}$ and $(SM)\overline{L}$. Senders are thus unlinked from their messages and receivers. Senders and messages are unlinked on all channels (content, length, timings) by using `EncToMix`, `pad` and `InboxMix`. Thus $(SM)\overline{L}$ is reached. The additional `SameAddr` ensures that the contacted mix is not receiver specific (as the primitive uses the same mix for all communications) and `Delay` ensures that the timing of sending observations at the senders does not correspond to their real intended time to send a message. Both the hidden sending timing (`Delay`) and the unlinking from senders and receivers (`SameAddr`) in combination with the other techniques lead us to $SM\overline{L}$.

$$\text{Null} \xrightarrow{\texttt{InboxMix+EncToMix+pad}} (SM)\overline{L}$$

$$(SM)\overline{L} \xrightarrow{\texttt{SameAddr+Delay}} SM\overline{L}$$

On the receiver and message side, basic mixnets achieve Null. Notice that indeed the message plaintext is sent over the final link to the receiver. So, the global adversary learns who received which message.

[15]Note that as there is no receiver or sender specific mix, it effectively implements both primitives.

A.9.2 DC-Nets

Description. DC-Nets [40] aim for the protection of both the actual senders and receivers against a GPA that additionally can corrupt all, but two participants. Therefore, in DC-Nets any user establishes symmetric keys with any other user and uses it for secret sharing: In each round each user either picks a real or a zero message and XORs it with all owned symmetric keys. The result is distributed to all participants. XORing the results of all users cancels out the symmetric keys (as each of them is used in two messages) and reveals the real message, or a collision if two or more real messages have been sent. To avoid or prevent collisions usually the mapping of sender per round is predetermined. The described protocol sends the real message to all users. If a message is intended for a specific receiver, implicit addressing can be used: the (key-private) encryption with the public key of the receiver replaces the "real message" above.

Translation to Primitives. *Fixed Message length and implicit addressing.* Messages are assumed to have a fixed length, which has the same effect as `pad`. Messages further are either learned in clear by all users (and even eavesdroppers on the links) or, if implicit addressing is used, the messages are end to end encrypted with a key-private scheme; implementing our primitive `E2EC(kp)`.

Broadcasting to receivers. As the XORing results and thus the message are delivered to all users in an order independent of the intended receiver, the primitive `Broadcast` is employed.

Sending order based on collision avoidance scheme. The order of senders given by the collision avoidance scheme is randomly chosen and thus implements our `Delay` primitive.

Mapping Secret Sharing. Secret sharing is more difficult to map to primitives. Arguably every sender's local result contributes to the final message of each round and is as large as the real message. Thus, it intuitively implements `reverseBroadcast`.

However, this only holds if collisions are completely prevented. If 2 or more real messages are (attempted to be) sent in a round, `reverseBroadcast` requires 2 or more sent messages per sender and thus the sender's local results (= 1 sent message) are no longer enough. Concretely, the occurrence of a collision leaks whether two messages are sent by the same or different senders. In the case of possible collisions, we rather understand secret sharing as a complex primitive combination: By XORing the real/zero message, we have `EncToMix`; by the homomorph properties of XOR and the fixed rounds, we have `InboxMix`, by the collection of all local results at one place (at every user), we have `SameAddr`, by the leakage due to collisions, we have `leakS` and finally by the zero message, we have a form of `D2S`.

The variant of `D2S` depends on whether we assume an ensured system maximum. If the maximum of 1 message being attempted to be sent per round is ensured by the DC-Net, i.e. collisions are completely prevented, it corresponds to $\texttt{D2S}_{\texttt{System}}$. Without any prerequisite, we can only conclude `AllSend` as it ensures that every user sends at least once, but there might be more attempted sendings.

Notions. While applying the identified primitives, we use `leakyDecMix` short for the combination of `Delay` + `EncToMix`+`pad`+`SameAddr`+`AnyMix`+`leakS`.

On the sender side without complete collision prevention, we reach $S\overline{O} - P$, i.e. the sender behavior is hidden completely, except for the grouping of messages per sender. Thus the adversary can tell for (some) pairs of messages whether they belong to the same sender or not, but not who this sender is/the senders are.

By using `leakyDecMix`, we unlink senders from messages and receivers in all dimensions (time, address, content), but still leak which messages are sent by the same user, but not by whom (corresponding to $SM\overline{L} - P$). By adding `AllSend`, i.e. ensuring that every sender sends at least once, it hides which senders are active (corresponding to $S\overline{O} - P$).

$$null \xrightarrow{\texttt{leakyDecMix}} S\overline{M}\overline{L} - P \xrightarrow{\texttt{AllSend}} S\overline{O} - P$$

On the sender side with complete collision prevention, we found two ways to $S\overline{O}$, i.e. the sender behavior is hidden completely. By using `reverseBroadcast` all information about the senders is removed as they are made equal in both batches (corresponding to $S\overline{O}$). As the complete collision prevention removes the leak, the other way starts with `DecMix` that unlinks senders from messages and receivers without additional restrictions (corresponding to $S\overline{M}\overline{L}$). To represent secret sharing, `DecMix` is combined with $\texttt{D2S}_{\texttt{System}}$ that perfectly hides the sending frequency and activity (corresponding to $S\overline{O}$) by ensuring that every sender has enough sending events to potentially have sent all real messages that the system can deliver.

$$null \xrightarrow{\texttt{reverseBroadcast}} S\overline{O}$$

$$null \xrightarrow{\texttt{DecMix}} S\overline{M}\overline{L} \xrightarrow{\texttt{D2S}_{\texttt{System}}} S\overline{O}$$

On the receiver side, by using `Broadcast` we reach $R\overline{O}$, i.e. the receiver behavior is completely hidden. Note however that without implicit addressing the receiver notion does not have much meaning as the adversary of course knows that the message is delivered and meant for all receivers.

$$null \xrightarrow{\texttt{Broadcast}} R\overline{O}$$

With implicit addressing, we even reach a notion protecting the combination of sender, message and receiver: $R\overline{O}\{S\overline{O} - P\}$, i.e. the receiver behavior, as well as the sent message and the sender behavior, except for the grouping of messages per sender are hidden.

With `Broadcast` we remove all information about the receivers (corresponding to $R\overline{O}$). With the encryption `E2EC(kp)` and padding `pad`, we hide the content (corresponding to $R\overline{O}[M\overline{O} - |m|]$) and also the length of the message (corresponding to $R\overline{O}[M\overline{O}]$). By additionally randomizing the sending order, but still leaking which communications belong to the same sender (`Delay+leakS`), we reach $R\overline{O}\{S\overline{M}\overline{L} - P\}$. Finally, by ensuring that all users sent at least once (`AllSend`) we reach $R\overline{O}\{S\overline{O} - P\}$.

$$null \xrightarrow{\texttt{Broadcast}} R\overline{O} \xrightarrow{\texttt{Enc(kp)}} R\overline{O}[M\overline{O} - |m|] \xrightarrow{\texttt{pad}} R\overline{O}[M\overline{O}]$$

$$R\overline{O}[M\overline{O}] \xrightarrow{\texttt{Delay+leakS}} R\overline{O}\{S\overline{M}\overline{L} - P\} \xrightarrow{\texttt{AllSend}} R\overline{O}\{S\overline{O} - P\}$$

Discussion. We now compare our derived notions with existing analysis results on DC-Nets.

Mapping the receiver side to `Broadcast` is intuitive and the reached notion is $R\overline{O}$.

On the sender side, Hevia and Micciancio [82] concluded that DC-Nets achieve their SA (Strong Anonymity), which translates to our $S\overline{O}$ (*Sender Unobservability*). Later Gelernter and Herzberg [72] show that based on collisions SA is not achieved, but only their new relaxed version $R_{SA}^{H,\tau}$, which translates to our $S\overline{O} - P$ (*Sender Unobservability leaking pseudonym*).

Under the assumption of complete collision prevention, both using `reverseBroadcast` and the combination of $\texttt{D2S}_{\texttt{System}}$ with `DecMix` result in $S\overline{O}$, which corresponds to the result of Hevia and Micciancio, as well as the fact that Gelernter and Herzberg's weaker notion is due to occurring collisions. Including collisions, our primitives conform to the result of Gelernter and Herzberg.

A.9.3 Pung

Description. Pung works in the client server model and wants to protect the behavior of the users against a corrupted server. Users exchange messages in Pung via dropping and retrieving them from post boxes at the Pung server. Communication partners learn each other's public keys and use them to derive a secret common key for authenticated end-to-end encryption, as well as the randomized labels to identify the post boxes they are going to use to send/retrieve messages to/from their partner in each round. Senders send their encrypted message and the label to the Pung server. Receivers use a Computationally Private Information Retrieval (CPIR) scheme to query the server once each round: Receivers calculate the label and craft an encrypted query vector that contains homomorphically encrypted 0 entries except for the position that corresponds to the label. The Pung server uses this vector to calculate an encrypted overlay of the query vector and the messages it stores. Thereby all but the requested message cancel out, without leaking information on the requested post box. Finally, Pung ensures that users send and receive exactly once per round. In case users are idle an encrypted dummy message to a random label is sent, as well as a the message from a random label retrieved.

Translation to Primitives. *Protecting the messages.* Pung uses symmetric, authenticated end-to-end message encryption and padding for all messages. This maps trivially to the primitives `E2EC(kp)` and `pad`.

Dummy Traffic. Users send and receive exactly once per round. This corresponds to $\texttt{D2S}_{\texttt{User}}$ and $\texttt{DfS}_{\texttt{User}}$ with a maximum of one message each.

Mapping CPIR. We assume that sending and receiving times are defined by Pung independent of users' intention to really send or receive. This translates to `Delay` for hiding the order of senders, as well as `OutboxMix` for hiding the timing relation between sending and receiving. The CPIR scheme ensures that the adversary cannot learn which post box was accessed, i.e. *all* communications are mixed (`SameAddr`) and that outgoing messages (from the Pung server) are a new encryption compared to the ingoing ones (`EncFromMix`).

Notions. While applying the identified primitives, we use `DecMix` short for the combination of `Delay` + `EncToMix`+`pad`+`SameAddr`+`AnyMix`.

Using multiple steps, Pung reaches Communication Unobservability ($C\overline{O}$), the strongest notion that completely hides messages, sender and receiver behavior, as well as even the fact whether real communications are happening under the given adversary model (especially including trust in communication partners).

With the help of encryption and padding, all information about the messages are hidden (corresponding to $M\overline{O}$) as in the example before. `DecMix` on top of an already encrypted message ensures the complete unlinking of senders from messages and receivers, as well as receivers from messages and senders. Notice that in this case the message is encrypted once for the receiver and once for the mix, as known from onion routing. Further by ensuring that every user has the same number of receiving events on top of the already existing protection, we additionally hide receiving activities and frequencies. With this, in addition to the protection for messages and senders, finally all receiver information is hidden (corresponding to $R\overline{O}\{SM\overline{L}\}$). With $\texttt{D2S}_{\texttt{User}}$ not only the sending activities and frequencies, but also the total number of real communications in the system is hidden (corresponding to $C\overline{O}$).

$$null \xrightarrow{\texttt{E2EC(kp)}} M\overline{O} - |m| \xrightarrow{\texttt{pad}} M\overline{O} \xrightarrow{\texttt{DecMix}} M\overline{O}[M\overline{L}] \xrightarrow{\texttt{DfS}_{\texttt{User}}} R\overline{O}\{SM\overline{L}\} \xrightarrow{\texttt{DS}_{\texttt{User}}} C\overline{O}$$

Discussion. Communication Unobservability ($C\overline{O}$) is even stronger than the notion that the authors stated for their system: Relationship Unobservability as informally defined by Pfitzmann and Hansen [126]. With $C\overline{O}$ no meaningful non-trivial information about *users* can be derived and

$C\overline{O}$ additionally requires to hide the number of real communications, which Pung does against the GPA with fixed communication times.

A.9.4 Loopix

Description. Loopix [128] employs mixes and additionally ingress and egress providers to protect sender and receiver privacy against a range of adversary models. They consider the GPA with extensions to i) perform network attacks, ii) corrupt mixes (eavesdrop or manipulate traffic), iii) corrupt providers (eavesdrop) and iv) corrupt users.

Every message is first sent to the sender's ingress provider and then through a layered mixnet to the receiver's egress provider. The egress provider collects messages for its users in postboxes (also in case they are offline). The routing path and the delay at each hop are chosen by the senders randomly for every message. The routing information (including the receiver's ID) is hidden at every hop with the Sphinx mix package format except for the locally required information, and the message is end-to-end encrypted.

Sending times are chosen randomly according to a fixed distribution. If the sender has no real message at the next sending time, she generates and sends a dummy message to a random egress provider, who later drops this message. Following a similar distribution, senders and mixes send loop dummy messages, i.e. messages addressed to themselves, to detect traffic manipulation via the absence of returning loop messages.

Receivers query their egress provider for received messages with a fixed frequency. Providers always respond with a fixed amount of messages by adding dummy messages if too few messages were received.

Translation to Primitives. *Messages.* All messages are key-private end-to-end encrypted and due to the Sphinx message format [52] padded to the same length, corresponding to the primitive `E2EC(kp)` and `pad`.

Mixnet. The layered mixnet with the Sphinx packet format ensures a different encryption of the message before and after every intermediate node, this directly corresponds to the primitive `EncToMix` and combined with the usage of end-to-end encryption also has the same effect as `EncFromMix`. Further, the shared layered mixnet for *all* communications corresponds to the primitive `SameAddr` and the randomly chosen per-hop delays correspond to the primitives `InboxMix` and `OutboxMix`.[16]

Sending Times and Dummy Traffic. The randomly chosen sending times correspond to `Delay`.

Senders pad their output to a fixed distribution using dummy messages, as well as receivers' inputs are padded to a fixed number with dummy messages. Thus the send events do not depend on the number of real sent messages. Intuitively, this is close to $\texttt{D2S}_{\texttt{User}}$. However, the dummy messages do not ensure that every sender sends a fixed number of times, but just pad according to a distribution with messages that end at random providers. Thus the actual protection is a bit weaker than what $\texttt{D2S}_{\texttt{User}}$ provides for special cases (and needs to be investigated in future work). One such special case is if Alice's random choice according to the distribution resulted in a low number of sent messages in this time interval. If we learn that more messages arrived in the corresponding interval that all belong to the same sender, we learn that Alice was not the sender. The likelihood of this depends on the chosen maximum and parameters of the distribution.

The receiver side intuitively seems to implement $\texttt{DfS}_{\texttt{User}}$, but on closer inspection Loopix does not implement any `DfS` primitive. As sender generated dummies are sent to random providers, we can estimate how many dummy messages arrive at a provider and know that the remaining messages are likely real messages. So, the number of messages is not padded to a maximum as $\texttt{DfS}_{\texttt{User}}$

[16]Note that if `SameAddr` is used, the mixing effect of `InboxMix`, `OutboxMix` or `InboxMix` and `OutboxMix` is the same as all communications are mixed with each other in any option.

requires, but instead is increased on average by an expected amount. The provider generated dummies cannot compensate for this as the critical adversarial observation happens before the provider.

Notions. We use `DecMix` short for `Delay` + `EncToMix`+`pad`+`SameAddr`+`InboxMix` and `EncMix` short for `EncFromMix` + `Pad` + `SameAddr`+`OutboxMix`:

On the sender side we see the unlinking of the sender from the message and receiver (in all dimensions) with the help of `DecMix` as in the examples before:

$$null \xrightarrow{\texttt{DecMix}} SM\overline{L}$$

On the receiver side we see the unlinking of the receiver from the sender and message (in all dimensions) similarly with `EncMix` :

$$null \xrightarrow{\texttt{EncMix}} RM\overline{L}$$

Further, the messages are completely protected (unless the receiver is corrupt) with the help of encryption and padding as in the examples before:

$$null \xrightarrow{\texttt{E2EC(kp)}} M\overline{O} - |m| \xrightarrow{\texttt{pad}} M\overline{O}$$

Discussion. As discussed earlier in Section 3.6.2, Loopix defines the goals of Sender-Receiver Third-party Unlinkability and Sender Online-/ Receiver Unobservability. Sender-Receiver Third-party Unlinkability is targeted against any of Loopix' adversary models, especially also insiders. Sender Online-/ Receiver Unobservability against any, except for insiders. Recall that Loopix' goal definitions are not completely formal and thus open to interpretation. In Section 3.6.2, we however argued that our understanding of Sender-Receiver Third-party Unlinkability is mapped to $(SR)\overline{L}$ and Sender Online-/Receiver Unobservability either both to $C\overline{O}$ or to $S\overline{O}$ and $R\overline{O}$.

Loopix' *Unobservability* is not backed by our primitives and indeed below we show that two types of frequency information can be derived from adversarial observations: 1) A corrupted provider (or receiver) can learn whether there is a large inbalance between the sending frequencies or not, as in the first case less messages are delivered to her than in the other. 2) A GPA can distinguish receiving frequencies of users at different providers as the dummy is just adding an expected value on each provider, not padding the number to a maximum.

Attack 1: Sender Frequency Attack on $SF\overline{L}$ against corrupted receiver/provider

We pick two senders A and B and a high number of messages k. In scenario 0 one message is sent by A, all other by B, in scenario 1 half of the messages are sent by B, the other half by A. The adversary polls messages from the provider in every time step and observes the number of delivered (real) messages.

The adversary chooses much more messages in the scenarios than can be sent until time point t where she evaluates her observations. In Loopix real messages (or dummy messages if no real messages are available) are sent following an exponential distribution with parameter $\frac{1}{\lambda}$. Thus, at point t in scenario 0 the adversary expects to observe $\lambda t + 1$ (one message from A and λt from B) and in scenario 1, $2\lambda t$ received messages. She guesses on the scenario where the real number of received messages is closer to the expectation. As the adversary can pick any number of messages and wait arbitrary long, the observations approach the expected values (law of large numbers) and thus with increasing t this strategy is approaching the optimal advantage.

Remark. Note that the same attack idea works if we limit the number of messages that are intended to be sent per time interval (the relaxed version of $S\overline{O}$ discussed in Section 4.4.1) unless the limit and chosen λ parameter ensure an overwhelming probability of all messages that are intended to be sent until this time are also being sent (enough sending events are chosen by the exponential distribution).

Remark 2. Notice that the same strategy works with a corrupted provider instead of receiver as the provider adds dummy messages per receiver and the number of expected loop messages is known. Thus the provider can estimate the number of real messages per receiver.

Attack 2: Receiver Frequency Attack on $RF\overline{L}-P'$ against GPA

We pick two receivers A and B and a high number of messages k. In scenario 0 one message is received by A, all other by B, in scenario 1 the single message is received by B, all other by A. The adversary counts how many messages enter A's and B's provider. If more messages enter A's provider than B's, the adversary guesses on scenario 1. If more messages enter B's than A's provider, the adversary guesses scenario 0.

As dummy traffic is addressed to a random provider the expected amount of dummy messages entering a provider is the same for all. As the adversary can make use of a large number of messages it follows that the number of dummy messages is close to the expected value and thus differences in the number of messages per provider are caused by real communications. Hence, the adversary's strategy is approaching the optimal advantage with increasing numbers of messages k.

B. Improving Bounds

Table B.1: Comparison of notations, N.A.: not applicable (concept does not exist/does not apply), -: no defined symbol

Our	**Trilemma**	**Counting-Bound**	**Optimality-Bound**	**Dropping-Bound**
$\mathcal{U}$, u_i	$\mathcal{S}$, u_i	$[n], p_i$ or S_i	P_i	-
$\mathcal{U}_H$	N.A.	H	N.A.	-
n	N	n	n	N
h	-	h	N.A.	-
c_a	N.A.	N.A.	N.A.	κN
c_p	c	N.A.	N.A	N.A.
l_{max}	$l+1$	N.A	N.A.	N.A.
r	N.A.	R	N.A.	N.A
$Out(r)$	$\approx p' \cdot r \cdot N$	$Out^{\pi}_{\sigma,R}$	-	-
p	$p = p' + \beta$	$\approx \frac{L^{\pi}_{i}(\sigma,R)}{R}$	$\sum_{j\in[n]} \lvert m_{i,j}\rvert$	N.A.
β	β	$\frac{Com^{\pi}_{\sigma,R} - Out^{\pi}_{\sigma,R}}{N*R}$	$l_i = \mu_{\mathbb{N}} - \sum_{j\in[n]} \lvert m_{i,j}\rvert$ $\approx \texttt{ovh}(T)/n = \mu_{\mathbb{N}}$	N.A.
σ_0, σ_1	N.A.	σ_0, σ_1	$\bar{M}^{(0)}$, $\bar{M}^{(1)}$	σ^0, σ^1
δ	δ	$Adv^{Comp-\mathbb{N}}_{\pi,n,A,Cap}(k)$	$Adv^{\mathbb{N}-annon}_{\pi,A}(k)$	$\mathsf{Adv}^{\Pi,A}(\sum, \mathsf{kickoff}, \mathsf{freeze})$
λ	η ($\delta \leq neg(\eta)$)	k ($Adv \leq negl(k)$)	k	λ
b	b	b	b	b

Table B.2: Overview used parameters

Parameter	**Meaning**
$\mathcal{U}$	set of senders
u_i	i^{th} sender
$\mathcal{U}_H$	set of honest senders
n	number of nodes/participants
h	number of honest senders, $\lvert\mathcal{U}_H\rvert$
c_a	number of actively compromised nodes
c_p	number of passively compromised (intermediate) nodes
l_{max}	latency, maximal delay of a message, the maximal number of rounds between the sending of a message and its reception
l_{exp}	expected delay of a message, average number of rounds between the sending of a message and its reception
r	number of rounds a certain metric or analysis refers to
$Out(r)$	delivered messages until round r
p'	probability of one node to send a real message in a given round
p	$p = p' + \beta$, probability to send any type of message
β	bandwidth-overhead, probability of one node to send a dummy message in a given round
σ_0, σ_1	scenarios
δ	adversary advantage
λ	security parameter
b	challenge bit

B.1. Tightening the claims

First, we make the effects of assumptions explicit by incorporating them into the analyzed dimensions. Thereby, we do not technically change any result, but allow to understand the real strength of the results better. Secondly, by in-depth analysis of the proofs, we found that the proofs work for even weaker assumptions than those that had been made, and in one case, we improved the calculations for the overhead.

B.1.1 Adversary Models

All papers assume global eavesdropping capabilities. However, as the actual attacks consider only one or two victim senders, we can reduce the global adversary to be local. She is limited to the links[1] of the victim(s).

Optimality-Bound, Counting-Bound and Trilemma

For these attacks the adversary only has to be able to notice when or how often the victim sends. In the integrated system model she has thus to be able to distinguish sending events from forwarding events. As a technicality the adversary in the proofs can decide that the victims do not receive any message. As thus all inbound packets must be followed by forwarding events, the adversary learns the number of real sending events by subtracting the outgoing packets from the inbound ones.

Dropping-Bound

For the bound the only active adversarial capacity needed is dropping, although their attacker model states multiple active capabilities (delay, create, modify and drop messages).

B.1.2 Privacy Notions Specification

Single Setting. For reasons of compatibility with the analyzed papers, we extend our notions from Chapter 3 by introducing an X_1 for each notion X. It expresses that every sender sends exactly once in each batch for a sender notion ($S\bar{O}$, $(SM)\bar{L}$), each receiver receives exactly once for a receiver notion ($R\bar{O}$), and each sender and receiver send/receive exactly once in each batch for an impartial notion ($C\bar{O}$, $(SR)\bar{L}$).

Formally, the extension X_1 is defined to any notion X as for any

sender notion: for all $b \in \{0,1\}, (u,q) \in Q_b : q = 1$, i.e. all users send exactly once in the batch.

receiver notion: for all $b \in \{0,1\}, (u,q) \in Q'_b : q = 1$, i.e. all users receive exactly once in the batch.

impartial notion: for all $b \in \{0,1\}, (u,q) \in Q_b \cup Q'_b : q = 1$, i.e. all users send and receive exactly once in the batch.

Note that this only expresses weaker privacy goals and in terms of bounds this means, the bound is also valid for the goal without this extension, but slightly less precise.

[1]This can be achieved trivially by their ISP, and probably easily by an attacking insider, who controls the nodes that are connected by the adjacent links.

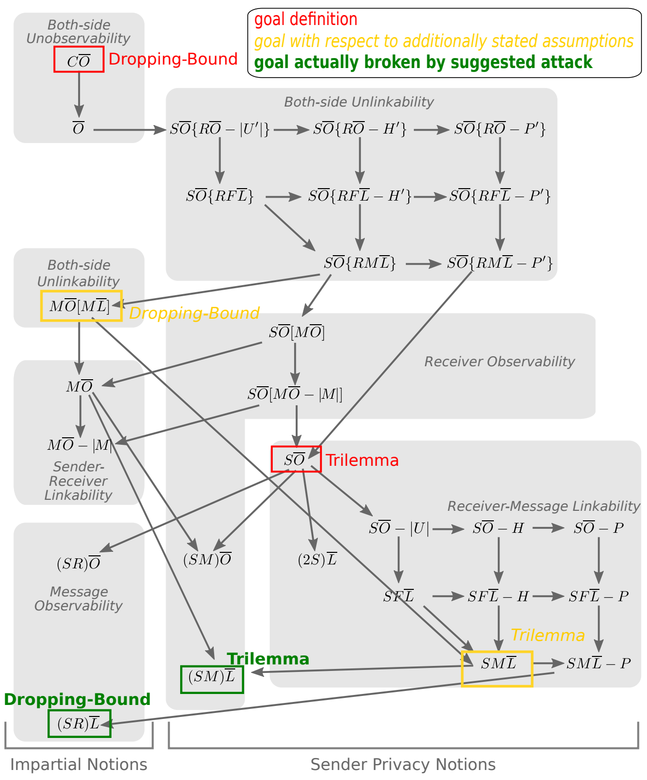
B.1.3 Dropping-Bound – Privacy Notion

The Dropping-Bound defines its own privacy goal without relation to other work. In its anonymity definition the game adversary is[2] not restricted in how she chooses the scenarios. Therefore, the described notion matches Communication Unobservability $C\overline{O}$, the strongest notion in the hierarchy.

Additionally to the anonymity definition the goal is however restricted by the "simple I/O setting", i.e. each participant sends and receives exactly one message. This restriction is equivalent to fixing the number of sending and receiving events to 1, which is the exact definition of $M\bar{O}[M\bar{L}]_1$, an already weaker impartial notion of the hierarchy.

The attack used in the proof breaks an even weaker notion. As it ignores message contents, we can use the same message in all communications. Further, we can define the second scenario equal to the first, with only the one sender u_0 that sends to the observed receiver switched with the alternative sender u_1 that sends to another receiver. Thereby, only the linking between those senders and receivers differs and the definition of $(SR)\bar{L}_1$ is met.

Interestingly, this is one of the weakest notions in the hierarchy. The bound is thus much stronger than the anonymity definition suggested (the strongest notion in the hierarchy), as their calculated cost is not only necessary to achieve a very strong privacy definition, but also if only the linking between sender and receiver is aimed to be protected (see Figure B.1).



goal definition
goal with respect to additionally stated assumptions
goal actually broken by suggested attack
Both-side Unobservability
$C\overline{O}$
Dropping-Bound
$\overline{O}$
Both-side Unlinkability
$S\overline{O}\{R\overline{O}-|U'|\}$
$S\overline{O}\{R\overline{O}-H'\}$
$S\overline{O}\{R\overline{O}-P'\}$
$S\overline{O}\{RF\overline{L}\}$
$S\overline{O}\{RF\overline{L}-H'\}$
$S\overline{O}\{RF\overline{L}-P'\}$
$S\overline{O}\{RM\overline{L}\}$
$S\overline{O}\{RM\overline{L}-P'\}$
Both-side Unlinkability
$M\overline{O}[M\overline{L}]$
Dropping-Bound
$S\overline{O}[M\overline{O}]$
Receiver Observability
$M\overline{O}$
$S\overline{O}[M\overline{O}-|M|]$
$M\overline{O}-|M|$
Sender-Receiver Linkability
$S\overline{O}$
Trilemma
Receiver-Message Linkability
$S\overline{O}-|U|$
$S\overline{O}-H$
$S\overline{O}-P$
$(SR)\overline{O}$
$(SM)\overline{O}$
$(2S)\overline{L}$
Message Observability
$SF\overline{L}$
$SF\overline{L}-H$
$SF\overline{L}-P$
Trilemma
Trilemma
$(SM)\overline{L}$
$SM\overline{L}$
$SM\overline{L}-P$
Dropping-Bound
$(SR)\overline{L}$
Impartial Notions
Sender Privacy Notions

Figure B.1: The mapping of the anonymity notions to our hierarchy of privacy notions from Section 3.5 (Figure 3.3): The notions used for the bound differ from the anonymity definition given due to additional assumptions. Further, the notions needed in the proofs are even weaker than the notions that follow from the additional assumptions. For a simplified summarizing presentation, we neglect the additionally restriction (X_1) that the Trilemma and the Dropping-Bound introduce for the notions $MO[M\overline{L}], SM\overline{L}, (SM)\overline{L}$ and $(SR)\overline{L}$.

[2]except for the behavior of corrupted users that does not hinder our comparison, as we discuss separately in Section 4.2.5.

B.1.4 Trilemma – Overview

We discuss the used privacy notion and tighten the bound in terms of the needed overhead.

Privacy Notion

The Trilemma uses sender anonymity from AnoA, which maps directly to $S\overline{O}$. This means that the two scenarios can arbitrarily differ in the senders, but in nothing else.

For the synchronous user distribution it is however additionally assumed that everyone can only send exactly one real message. Similarly to the Dropping-Bound before, this means that the frequency with which a sender sends needs not to be hidden, as it is identical in both scenarios. This is equal to the definition of $SM\overline{L}_1$.

The analyzed notion in the proof of the synchronized model, as opposed to the claimed goal or the goal that follows from the stated assumptions, changes however only the sender of the challenge message (with some other sender[3]). This matches the definition of $(SM)\bar{L}_1$ as only the connection between two senders and messages is changed and every sender sends one message.

In the unsynchronized model, it is assumed that each user wants to send messages. This time the number of messages is not fixed by the notion, but neither can it be chosen by the adversary. Every time the coin flip decides that the user has to send a message, she is assumed to have one ready to send. As again the only difference allowed is the change of senders for the challenge message, this translates to $(SM)\bar{L}$.

Similarly to the Dropping-Bound, we see that even though a pretty strong notion was stated in the beginning, the suggested attack breaks one of the weakest notions defined; the notion that only protects the linking between sender and message, but keeps anything else identical (see Figure B.1).

Bound

The idea of the proof in the unsynchronized case is simple: An adversary knows that the sender of the critical message has sent in the $l_{max} - 1$ rounds before she received this message. Thus, if one of the two victim senders did not sent in these rounds, we know the other must have been the sender, as the only uncertainty the adversary has left is which of those two candidate users was the sender. Therefore, the adversary wins, i.e. learns the sender, if the alternative sender did not send.

The authors perform intricate calculations, introducing random variables, the Chernoff bound and Markov's inequality, to prove their bound:

$$\delta \geq 1 - \left[\frac{1}{2} + min\left(\frac{1}{2}, 1 - (1-p)^{l_{max}-1}\right)\right]$$

Which is equivalent to

$$\delta \geq \frac{1}{2} - min\left(\frac{1}{2}, 1 - (1-p)^{l_{max}-1}\right)$$

and can be even easier understood as:

$$\delta \geq \max\left(0, (1-p)^{l_{max}-1} - \frac{1}{2}\right)$$

[3]This is not made explicit, but has to be done to respect the assumption of every sender sending exactly one message.

However, considering that we only need to bound the probability that the other user does not send, we claim that an easier and more accurate bound is:

$$\delta \geq (1-p)^{l_{max}-1}$$

We know that with probability $1-p$ the alternative user does not send in one round. As the sending in the rounds are independent (as stated in [54]), $(1-p)^{l_{max}-1}$ is the probability that the user does not send in any of the rounds.

With this difference, we adapt the argumentation of [54] (which we explain intuitively together with the other bounds in Section 4.2.7) for the extended case with compromised protocol parties as well and result in (cf. Appendix B.2.3):

$$\delta \geq \begin{cases} 1 - \left[1 - \binom{c}{l_{max}-1} / \binom{K}{l_{max}-1}\right] & \left[1-(1-p)^{l_{max}-1}\right] \\ & \text{if } c_p \geq l_{max}-1 \\ \left(1 - \left[1 - 1/\binom{K}{c_p}\right]\left[1-(1-p)^{c_p}\right]\right) & \left((1-p)^{l_{max}-1-c_p}\right) \\ & \text{if } c_p < l_{max}-1 \end{cases}$$

B.1.5 Trilemma – Compromising adversary

Extending the adversary to compromise up to $c_p \leq n-2$ intermediate nodes facilitates the attack of tracing messages along their anonymization paths, if all nodes on these paths are under adversarial control. This increases the advantage of the adversary, and the Trilemma is interested in this additional probability for an attack to succeed. We use K to denote the number of protocol parties throughout this section and discuss the *synchronized user setting* in the following.

Recall that the bound for synchronized users without corruption is:

$$\delta \geq 1 - min\left(1, \frac{(l_{max}-1)(1+\beta n)}{n-1}\right)$$

According to [54] we define the last part (the probability that a certain user has sent a message in the $l_{\max}-1$ rounds) to be

$$f_\beta(l_{\max}-1) := min\left(1, \frac{(l_{max}-1)(1+\beta n)}{n-1}\right).$$

For corrupted intermediaries, [54] distinguishes two cases. The adversary either has a chance to compromise all relays on the anonymization path of the challenge message as she has corrupted enough relays, or not. The authors simplify the first case and bound the probability that the challenge or alternative messages can be traced with the probability that all relays on the anonymization path are compromised: $\binom{c_p}{l_{max}-1}/\binom{K}{l_{max}-1}$. The adversary can only lose if some relay on the path is honest $\left(1 - \frac{\binom{c_p}{l_{max}-1}}{\binom{K}{l_{max}-1}}\right)$ and an alternative message is sent ($f_\beta(l_{\max}-1)$). She thus loses with a probability of at most $\left(1 - \frac{\binom{c_p}{l_{max}-1}}{\binom{K}{l_{max}-1}}\right) f_\beta(l_{\max}-1)$. As she wins in the complement to this event, her advantage in this case is at least:

$$1 - \left(1 - \frac{\binom{c_p}{l_{max}-1}}{\binom{K}{l_{max}-1}}\right) f_\beta(l_{\max}-1).$$

In the second case, not all intermediate nodes can be corrupted. Note that for the adversary to win it suffices to track all alternative messages until the challenge message is received (as she can exclude them). The adversary hence loses if an alternative message is sent and an honest relay is on the path that this message shares with the challenge message. There is an honest relay on this path if the message traversed more relays ($> c_p$) than the adversary can compromise ($f_\beta(l_{\max} - 1 - c_p)$). However, there might also be an honest relay on this path if the path is shorter (consisting of $\leq c_p$ relays). This event is at most as likely as having an honest relay in exactly c_p relays: $1 - 1/\binom{K}{c_p}$. As a shorter path occurs with probability $f_\beta(c_p)$, the adversary loses at most with the probability $f_\beta(l_{\max} - 1 - c_p) + f_\beta(c_p)(1 - 1/\binom{K}{c_p})$. The adversary wins in the complementary event, so her advantage is at least

$$1 - \left[1 - 1/\binom{K}{c_p}\right] f_\beta(c_p) - f_\beta(l_{\max} - 1 - c_p).$$

The two considerations result in the final bound:

$$\delta \geq \begin{cases} 1 - \left[1 - \binom{c_p}{l_{max}-1}/\binom{K}{l_{max}-1}\right] f_\beta(l_{\max} - 1) & c_p \geq l_{max} - 1 \\ 1 - \left[1 - 1/\binom{K}{c_p}\right] f_\beta(c_p) - f_\beta(l_{\max} - 1 - c_p) & c_p < l_{max} - 1 \end{cases}$$

For the *unsynchronized setting* the same ideas are applied on the basis of the non-compromising bound for the unsynchronized setting (cf. Appendix B.2.3).

As for the non-compromising case, the above bounds induce an *area of impossibility for the compromising adversary.* If an adversary passively compromises $c_p < l_{max} - 1$ protocol parties, then the area of impossibility is

$$2(l_{max} - 1 - c_p)\beta \leq 1 - \frac{1}{poly(\lambda)}.$$

If the number of compromised nodes is $c_p \geq l_{max} - 1$, then anonymity cannot be reached for

$$2(l_{max} - 1)\beta \leq 1 - \frac{1}{poly(\lambda)} \text{ and } l_{max} \in O(1).$$

B.2. Proofs

B.2.1 Advantage Definitions

All advantage definitions, i.e. how high the probability of the adversary guessing correctly has to be to break the privacy notion, are equivalent. We use $Pr[g = \mathcal{A}|\mathcal{C}(b)]$ short for the probability that the attacker $\mathcal{A}$ guesses g when the challenger $\mathcal{C}$ picked random bit b.

Counting-Bound. This definition requires the probability that any adversary algorithm $\mathcal{A}$ correctly guesses that bit $b = 1$ ($Pr[1 = \mathcal{A}|\mathcal{C}(1)]$), is only negligibly bigger than the same algorithm guessing $b = 1$ incorrectly ($Pr[1 = \mathcal{A}|\mathcal{C}(0)]$). So, the adversary has only a negligible advantage δ in winning the game.

$$Pr[1 = \mathcal{A}|\mathcal{C}(1)] - Pr[1 = \mathcal{A}|\mathcal{C}(0)] \leq \delta$$

Optimality-Bound. The definition of the adversary's attack advantage δ in the Optimality-Bound can be shown to be equivalent, under the assumption that the adversary always guesses something, with simple transformations:

$$\begin{aligned} \delta_{Counting-Bound} &= \Pr(1|1) - \Pr(1|0) \\ &= \Pr(1|1) - (1 - \Pr(0|0)) \\ &= 2 \cdot (0.5 \cdot \Pr(0|0) + 0.5 \cdot \Pr(1|1)) - 1 \\ &= 2 \cdot \Pr(b|b) - 1 = \delta_{Optimality-Bound} \end{aligned}$$

Trilemma. The Trilemma uses a definition similar to the Counting-Bound's as follows:

$$Pr[0 = \mathcal{A}|\mathcal{C}(0)] \leq Pr[0 = \mathcal{A}|\mathcal{C}(1)] + \delta$$

Note that in comparison to the Counting-Bound's here only the bits are changed. Technically, the advantage definition has to be fulfilled for any PPT adversary. Hence, if there is an adversary violating the definition of the Counting-Bound, we can simply swap its chosen scenarios and invert the output bit and we have an adversary violating the definition of the Trilemma and similarly for the other way round.

Dropping-Bound. This anonymity defines two scenarios to be indistinguishable iff the statistical distance between the observation of the adversary is negligible in the security parameter. As measure for the statistical distance the total variation distance ($\Delta_{TV}(\cdot, \cdot)$) is used.

$$\Delta_{TV}(V_{\mathcal{A}}|\mathcal{C}(0), V_{\mathcal{A}}|\mathcal{C}(1)) \leq \delta$$

From [50] (Equation 8) we know that this total variance based definition and the differential privacy based $(0, \delta)$-closeness definition of [50] are interchangeable. Further, the $(0, \delta)$-closeness definition is defined as the outputs of the mechanism, i.e. the input to the game adversary, being indistinguishable, just as the probabilities in the definition of the Counting-Bound and Trilemma: $V_{\mathcal{A}}|\mathcal{C}(0) \leq V_{\mathcal{A}}|\mathcal{C}(1) + \delta$.

Thus, the only remaining difference between the definitions is that the Dropping-Bound is using the probability distributions in the views of the game adversary, while the Counting-Boundis using all possible game adversary algorithms. However, if the difference of the probability distributions in the views is negligible, so is the chance of any adversary to distinguish them. Also, if there is an adversary that can distinguish the scenarios, then the probability distributions in the views have to be non-negligibly different.

B.2.2 Counting-Bound and Optimality-Bound Bound Equivalence

The Optimality-Bound proves optimality of a protocol adaption, precisely the adding of dummy traffic: The overhead `ovh` of each such protocol adaption τ has to ensure that each of the n possible senders is sending the maximum number of messages μ_{max}. This leads to $n \cdot \mu_{max}$ send events:

$$\texttt{ovh}(\tau) \geq n \cdot \mu_{max}$$

The Counting-Bound wants to prevent the adversary from excluding any sender from the set of suspects that could have sent *all* messages. Thus the number of real messages received can be at most the total number of messages (real and dummy) any one sender has sent.

More formally, let $Out(r)$ denote[4] the number of messages received by a destination until round r, $L_i(r)$ the number of messages sent by sender u_i and $\mathcal{U}_H$ the set of honest users. The bound is

$$Out(r) \leq \min\{L_i(r)|u_i \in \mathcal{U}_H\}.$$

It follows that the total number of sending events for all senders $Com(r)$ has to be sufficiently high:

$$Com(r) \geq Out(r) \cdot |\mathcal{U}_H| = Out(r) \cdot h$$

Comparison. As μ_{max} messages are delivered, $Out(r) = \mu_{max}$. The total number of messages sent are $Com(r) = ovh(\tau)$. Since there are no corrupted users, $h = n$. From this we conclude equality of the bounds :

$$\texttt{ovh}(\tau) \geq n \cdot \mu_{max} \iff Com(r) \geq n \cdot Out(r) = h \cdot Out(r)$$

[4]Compared to [72], we omit additional parameters. I.e. $Out(r)$ is short for $Out^{\pi}_{\sigma,r}$.

B.2.3 Improving the Trilemma

Case 1: $c_p \geq l_{max} - 1$ This means all intermediate nodes chosen in the $l_{\max} - 1$ rounds could be corrupted. As for the synchronous behavior, the attackers definitively wins if all intermediate nodes are corrupted ($\binom{c}{l_{max}-1}/\binom{K}{l_{max}-1}$). He also wins if the alternative user does not sent ($(1-p)^{l_{max}-1}$). So, her advantage can be bound by the complementary event to not all intermediate nodes being corrupted ($1 - \binom{c}{l_{max}-1}/\binom{K}{l_{max}-1}$) and the probability that the other user sends ($1-(1-p)^{l_{max}-1}$):

$$1 - \left[1 - \binom{c}{l_{max}-1}/\binom{K}{l_{max}-1}\right]\left[1-(1-p)^{l_{max}-1}\right]$$

Case 2: $c_p < l_{max} - 1$ This means not all intermediate nodes are corrupted. As for the synchronous behavior, the attacker wins except if an alternative and the challenge message share a long path (so long that an honest node has to be on it) $(1-(1-p)^{l_{max}-1-c_p})$ or there are *only* alternative messages that share short paths (and none that shares a long path)[5] $((1-p)^{l_{max}-1-c_p}(1-(1-p)^{c_p}))$ but an honest node is on it ($\leq 1 - 1/\binom{K}{c_p}$):

$$\begin{aligned}\delta &\geq 1 - \left(1-(1-p)^{l_{max}-1-c_p}\right) \\ &\quad - (1-p)^{l_{max}-1-c_p}\left[1-(1-p)^{c_p}\right]\left[1 - 1/\binom{K}{c_p}\right] \\ &= (1-p)^{l_{max}-1-c_p} - (1-p)^{l_{max}-1-c_p}\left[1-(1-p)^{c_p}\right]\left[1 - 1/\binom{K}{c_p}\right] \\ &= \left((1-p)^{l_{max}-1-c_p}\right)\left(1 - \left[1-(1-p)^{c_p}\right]\left[1 - 1/\binom{K}{c_p}\right]\right)\end{aligned}$$

B.2.4 Impossibility areas

Relations between variables The number of send messages $Com(r)$ are the dummy messages (β messages per user and round) and real messages that are delivered ($Out(r)$).

$$Com(r) = \beta nr + Out(r)$$

Transformation Using our discovered relation between the variables and the Trilemma's assumption that $n \approx poly(\lambda)$ we can transform the Counting-Bound:

$$\begin{aligned} Com(r) &\geq Out(r) \cdot n \\ \beta nr + Out(r) &\geq Out(r) \cdot poly(\lambda) \\ \beta &\geq \frac{Out(r) \cdot (poly(\lambda) - 1)}{poly(\lambda) \cdot r} \\ \beta &\geq \frac{Out(r)}{r} \cdot \left(1 - \frac{1}{poly(\lambda)}\right)\end{aligned}$$

and the Trilemma (impossibility area): $2(l_{max}-1)\beta \geq 1 - \frac{1}{poly(\lambda)} \iff \beta \geq \frac{1}{2(l_{max}-1)}\left(1 - \frac{1}{poly(\lambda)}\right)$

B.2.5 No latency in the Trilemma

Using $l_{max} = 1$ yields:

[5]Note that this is a tighter estimation as the one of synchronized user setting, where the probability of a short shared path($f_\beta(c_p)$) is used (and the existence of further alternative messages is neglected).

synchronized:	$\delta \geq 1 - min\left(1, \frac{0+\beta n0}{n-1}\right) = 1$
unsynchronized (original):	$\delta \geq 1 - \left[\frac{1}{2} + min\left(\frac{1}{2}, 1-(1-p)^0\right)\right] = \frac{1}{2}$
unsynchronized (improved):	$\delta \geq (1-p)^0 = 1$

B.3. Extended Results

B.3.1 Receiver Privacy Goals

Both the Optimality-Bound as well as the Trilemma also consider receiver privacy goals.

The analysis for the Optimality-Bound is consistent to its bound for $S\overline{O}$: if any user receives less than the total number of real messages, she is excluded from the anonymity set. Both sender and receiver bounds hence are equal, and $R\overline{O}$ can only be achieved with high bandwidth overhead; for instance, by implementing a broadcast.

The Trilemma also adapts its original attack to identify receivers: The adversary observes the sending of the challenge message and concludes that the message can only be received by someone who receives a message within the next l_{max} rounds. If enough relays are corrupted, the message can be traced. Interestingly, the resulting bound postulates a lower cost than for the senders. Attacking the sender, the candidate messages are only those sent within the l_{max} rounds before the challenge message is received. Attacking the receiver, the candidate set expands to those messages sent during the $l_{max}-1$ rounds before, and the $l_{max}-1$ rounds after sending the challenge message. All could have caused a message reception during the critical period, depending on how the protocol determines the latency for each message. We hypothesize that future work might improve this bound to match the sender case, because not even an optimal protocol can be able to ensure that all messages *always* end up being received in the critical period[6].

B.3.2 Note on related results

It is interesting to note, that researchers on the physical layer defined privacy goals that are similar, and identified the same bound as the Optimality- and the Counting-Bound [148]. Assuming the lack of a shared secret, they additionally analyze how much shared randomness is needed between the users.

Oya et al. analyze how a given (total) amount of dummy traffic should be spent on different dummy traffic kinds in pool mix networks to optimally improve their privacy [123]. While this is an interesting problem, it differs in two ways from the analyses we systematized in this work: First, it does not propose a bound on the number of dummy messages required to achieve a certain privacy goal, but shows how to best use a dummy traffic budget. Second, their privacy measure is the mean squared error between the *real probability* for a communication of a sender-receiver pair and the adversary's estimate. This is conceptually different from our game-based approaches, in which the adversary has to distinguish two *hypothetical worst-case* scenarios. While a larger mean squared error represents better privacy, it is not obvious which mean squared error is to be considered sufficient (as compared to the game-based notions that are either achieved or not achieved).

[6]As receiving all these messages in one l_{max} interval implies that less receive events occurred during the l_{max} interval before.

C. Details on Formal Onion Routing

C.1. Ideal Functionality

C.1.1 No replies

The following description stems from [31]. Adapted parts are highlighted. Additionally, we provide a pseudo code description as the black parts[1] of Algorithm 12 and 13.

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

Ideal Onion Routing Functionality: Data Structure.

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

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

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

1. If $|\mathcal{P}| \geq N$, reject.

[1] The teal parts are for the repliable case.

2. Otherwise, create a new session id sid randomly, and let $O = (sid, P_s, P_r, m, n, \mathcal{P}, 0)$.

3. If P_s is corrupted, send "start belongs to onion from P_s with $sid, P_r, m, n, \mathcal{P}$" to the adversary S. Send itself message $(Process\_Next\_Step, O)$.

$(Process\_Next\_Step, O)$. This is the core of the ideal protocol. Suppose $O = (sid, P_s, P_r, m, n, P, i)$. The ideal functionality looks at the next part of the path. The router P_{o_i} just processed[2] the onion and now it is being passed to $P_{o_{i+1}}$. Corresponding to which routers are honest, and which ones are adversarial, there are two possibilities for the next part of the path:

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

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

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

- P_{o_i} is the last honest router on the path; in particular, this means that P_r is adversarial as well. In that case, the ideal functionality sends the message "Onion from P_{o_i} with message m for P_r routed through $(P_{o_{i+1}}, ..., P_{o_n})$" to the adversary $\mathcal{S}$. If P_s is corrupted it further adds "(end) belongs to onion from P_s with $sid, P_r, m, n, \mathcal{P}$" to the message for S. (Note that if $P_{o_{i+1}} = P_r$, the list $(P_{o_{i+1}}, ..., P_{o_n})$ will be empty.)

(Deliver_Message, temp). This is a message that $\mathcal{S}$ sends to the ideal process to notify it that it agrees that the onion with temporary id $temp$ should be delivered to its current destination. To process this message, the functionality checks if the temporary identifier $temp$ corresponds to any onion O on the list L. If it does, it retrieves the corresponding record $(temp, O, j)$ and updates the onion: if $O = (sid, P_s, P_r, m, n, \mathcal{P}, i)$, it replaces i with j to indicate that we have reached the j'th router on the path of this onion. If $j < n+1$, generates a temporary identifier $temp'$, sends "Onion $temp'$ received" to party P_{o_j}, and stores the resulting pair $(temp', O = (sid, P_s, P_r, m, n, \mathcal{P}, j)$ in the buffer B_{o_j} of party P_{o_j}. Otherwise, $j = n+1$, so the onion has reached its destination: if $m \neq \perp$ it sends "Message m received" to router P_r; otherwise it does not deliver anything[3] .

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

- Checks if the temporary identifier $temp'$ corresponds to any entry in B_i. If it does, it retrieves the corresponding record $(temp', O)$.

- Sends itself the message $(Process\_Next\_Step, O)$.

- Removes $(temp', O)$ from B_i.

This concludes the description of the ideal functionality. We must now explain how the ideal honest routers work. When an honest router receives a message of the form "Onion $temp'$ received" from the ideal functionality, it notifies environment $\mathcal{Z}$ about it and awaits instructions for when to forward the onion $temp'$ to its next destination. When instructed by $\mathcal{Z}$, it sends the message "Forward Onion $temp'$" to the ideal functionality." [31]

[2]In case $i = 0$, processed means having originated the onion and submitted it to the ideal process.
[3] This is needed to account for the fact that the adversary inserts onions into the network that at some point do not decrypt correctly.

C.1.2 With Replies

We show the ideal functionality in Algorithm 12 and 13. As before, the honest nodes inform the environment about the *temp* whenever they receive an onion. Further, they now additionally include the information whether the onion is repliable. We highlight the changes compared to the ideal functionality of the simple one-way sending in teal. We especially add two data structures:

- *Back*: to store the mapping from *temp*s (labels of the onions) to the corresponding path and forward onion id. This mapping is used to find the right path when a reply onion is constructed.
- ID_{fwd}: to store the mapping from backward *id*s to forward *id*s. This mapping is used to allow corrupted senders (i.e. backward receivers) to learn all information of the backward onion; including to which forward onion she belongs.

We assume that a corrupted sender (i.e. backward receiver) can learn and link all onion layers. Further, an onion can be replied to multiple times. We stress that this is a useful security definition as single use reply blocks can be created with the help of duplicate protection (on the header), which also prevents other traffic analysis attacks that are considered an orthogonal problem.

C.1.3 Analyzing the Ideal Functionality without Replies

There indeed is confusion about which privacy the ideal functionality $\mathcal{F}$ of [31] actually guarantees. The work itself states only that "it's 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."

[8, 12, 129, 130, 153] state that this translates to the degree of anonymity Tor provides, although [57, 65] argue that it is not applicable for Tor. [13] states that it "hide(s) the source and destination over a network," [124] even understood it as "a concrete ZK proof of senders' knowledge of their messages" and [19] as "provable reduction from unlinkability to traffic analysis." [70] states that the privacy is "that an adversary cannot correctly guess relations between incoming messages and outgoing messages at onion routers, and [...] that each onion router cannot know the whole route path of any onion." While [65] and [64] realize that the anonymity is not analyzed and suspect it to be close to the one of [111], which claims to have sender and receiver anonymity against a global passive adversary [64].

Analysis under Restricted Adversary Model

Instantiation of our 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 3 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 is 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

Algorithm 12: Ideal Functionality $\mathcal{F}$ (Part 1)

```
Data structure:
Bad: Set of corrupted nodes
$L$: List of onions processed by adversarial nodes
$B_i$: List of onions held by node $P_i$
$Back$: Mapping from temps to path and forward id
$ID_{fwd}$: Mapping from backward id to forward id
// Notation:
// $\mathcal{S}$: Adversary (resp. Simulator)
// $\mathcal{Z}$: Environment
// $\mathcal{P} = (P_{o_1}, \dots, P_{o_n})$: Onion path, ($\mathcal{P}^{\rightarrow}$ forwards, $\mathcal{P}^{\leftarrow}$ backwards)
// $O = (id, P_s, P_r, m, \mathcal{P}, \mathcal{P}', i, d)$: Onion = (identifier, sender, receiver, message, path in
   current direction, path in other direction, traveled distance, direction)
// $N$: Maximal onion path length
On message Process_New_Onion($P_r, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}$) 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
        $id \leftarrow^R$ session ID ;                              // pick random session ID
        $O \leftarrow (id, P_s, P_r, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, 0, f)$ ;      // create new onion
        Output_Corrupt_Sender($P_s, id, P_r, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}$, start, $f$);
        Process_Next_Step($O$);
On message Process_New_Backward_Onion($m, temp$) from $P$
    // $P$ creates and sends a backward onion (either instructed by $\mathcal{Z}$ if honest or $\mathcal{S}$ if
       corrupted)
    if $Back(temp) = \perp$ ;                                      // no forward onion was sent
     then
        Reject;
    else
        $Back(temp) = (P_s, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, P_r, id')$;     // lookup the corresponding path
        $id \leftarrow^R$ session ID;                               // pick random session ID
        Store $id'$ under $ID_{fwd}(id)$;                          // add ID linking to mapping
        $O \leftarrow (id, P_r, P_s, m, \mathcal{P}^{\leftarrow}, \mathcal{P}^{\rightarrow}, 0, b)$;      // create new onion
        Output_Corrupt_Sender($P_r, id, P_s, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}$, start, $b$);
        Process_Next_Step($O$);
Procedure Output_Corrupt_Sender($P_s, id, P_r, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, temp, d$)
    // 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 $id, P_r, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, b$" to $\mathcal{S}$;
        if $d = b$ then
            add "as answer to $ID_{fwd}(id)$" to the output for $\mathcal{S}$
```

Algorithm 13: Ideal Functionality $\mathcal{F}$ (Part 2)

```
Procedure Process_Next_Step($O = (id, P_s, P_r, m, \mathcal{P}, \mathcal{P}'i, d)$)
    // 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 $temp$ in direction $d$ from $P_{o_i}$ with message $m$ for $P_r$ routed through
          $(P_{o_{i+1}}, \dots, P_{o_n})$" to $\mathcal{S}$;
        if $d = f$ then
            Store $(P_s, \mathcal{P}, \mathcal{P}', P_r, id)$ under $Back(temp)$;
            Add "$temp$'s first part of the backward path is $\mathcal{P}'_H$ " with $\mathcal{P}'_H$ being $\mathcal{P}'$ until (and
              including) the first honest node to the message for $\mathcal{S}$;
            Output_Corrupt_Sender($P_s, id, P_r, m, \mathcal{P}, \mathcal{P}', temp, f$);
        else
            Output_Corrupt_Sender($P_r, id, P_s, m, \mathcal{P}', \mathcal{P}, temp, b$);
    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;
        $temp \xleftarrow{R}$ temporary ID;
        Send "Onion $temp$ from $P_{o_i}$ routed through $(P_{o_{i+1}}, \dots, P_{o_{j-1}})$ to $P_{o_j}$" to $\mathcal{S}$;
        Add $(temp, O, j)$ to $L$;          // see Deliver_Message($temp$) to continue this routing
        if $d = f$ then
            Output_Corrupt_Sender($P_s, id, P_r, m, \mathcal{P}, \mathcal{P}', temp, f$);
        else
            Output_Corrupt_Sender($P_r, id, P_s, m, \mathcal{P}', \mathcal{P}, temp, b$);
            if $P_s \in Bad$ and $i = 0$ then
                Send "temp belongs to id" to $\mathcal{S}$;

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 $(temp, O = (sid, P_s, P_r, m, \mathcal{P}, \mathcal{P}', i), j)$ from $L$;
        $O \leftarrow (sid, P_s, P_r, m, \mathcal{P}, \mathcal{P}', j)$;                // $j$-th router reached
        if $j < |\mathcal{P}| + 1$ then
            $temp' \xleftarrow{R}$ 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 $m$ under $temp$ in direction $d$ received to $P_r$";
                if $\mathcal{P}' \neq ()$ and $d = f$ then
                    add "that is repliable" to the message for $P_r$;
                    Store $(P_s, \mathcal{P}, \mathcal{P}', P_r, id)$ under $Back(temp)$;

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($O$);
```

prohibit replay attacks, which we consider as a special kind of traffic analysis attack, the adversary class drops any duplicated deliver requests from the adversary.

Analysis

Recall that 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_{c^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 13, which we show below.
Theorem 13. *$\mathcal{F}$ achieves $M\overline{O}_{c^e}$, $SM\overline{L}_{c^s}$ and $RM\overline{L}_{c^0}$, and those implied by them, but no other notions of our hierarchy (Figure 3.3 of Section 3.5) for $\mathcal{C}$.*

Lemma 6. *$\mathcal{F}$ achieves $M\overline{O}_{c^e}$ for $\mathcal{C}$.*

Proof. We go through the messages of the ideal functionality $\mathcal{F}$ and check whether they help to distinguish two scenarios differing only in the messages of honest users.

Process_New_Onion: Does only output information to the adversary for corrupted senders, which will be equal ($m, \mathcal{P}$ etc.) or randomly generated (sid) in both scenarios because of X_{c^e}.

Process_Next_Step: Information output for corrupted senders is equal or random because of X_{c^e}. Hence, we can focus on honest senders. As corrupted receivers receive the same messages and everything else is equal in both scenarios, the adversary gets identical output for corrupted receivers. For honest receivers the adversary only gets messages "Onion *temp* from P_{o_i} routed through $(P_{o_{i+1}}, \ldots, P_{o_{j-1}}$ to $P_{o_j})$" or "Onion *temp* from P_{o_i} to $P_{o_{i+1}}$". Since everything except the messages is equal in both scenarios, the path is equal in both scenarios and does not help the adversary distinguish. Further *temp* is generated randomly by $\mathcal{F}$ and hence does not help the adversary distinguish.

Deliver_Message: Because of the adversary class $\mathcal{C}$ the attacker cannot exploit sending such messages.

Forward_Onion: Is a message between honest routers and the ideal functionality. Hence, the adversary cannot influence it or get information from it. □

Lemma 7. *$\mathcal{F}$ achieves $SM\overline{L}_{c^s}$ for $\mathcal{C}$.*

Proof. X_{c^s} excludes corrupted senders and hence, we can ignore outputs that happen for corrupted senders. $\mathcal{C}$ forbids the misuse of "Deliver_Message" and hence all onions went through the network by honest processing of the routers and no onions can be replayed or delayed.

Then the ideal functionality only outputs every part on the path between honest routers once, and if the receiver is corrupted the message once, for all communications that the adversary picked for the chosen scenario. $\mathcal{C}$ also guarantees that all paths share a common honest router. Let $\mathcal{P}_{tohonest}$ be the set of paths that lead to the honest router and $\mathcal{P}_{fromhonest}$ the set with paths that start from the honest router. Since $\mathcal{C}$ chooses at least one honest router, such that the maximum path length is met, i.e. any of the possible path combinations $\mathcal{P}_{tohonest} \times \mathcal{P}_{fromhonest}$ are shorter than N. Because of $\mathcal{C}$, the outputs will additionally be in mixed order and not linkable because of the order in which she observes them. No path combination can be excluded by the adversary, as all are valid paths, and hence she has no information that helps her deciding on the total path. Further, she only learns which receiver receives which message. Since, this is the only information she has and she cannot exclude any path, she cannot do better than to randomly guess the sender-receiver and hence sender-message pairs. □

Lemma 8. *$\mathcal{F}$ achieves $RM\overline{L}_{c^0}$ for $\mathcal{C}$.*

Proof. X_{c^0} excludes that the adversary learns different receiver-message combinations as outputs of $\mathcal{F}$ as the message is never output in this case. The only other option to distinguish the scenarios is to exploit that the adversary knows which message is sent by which sender. However, as argued in the proof of Lemma 7, it is not possible to link the parts of the path for $\mathcal{C}$. □

Lemma 9. *$\mathcal{F}$ does not achieve any notion of our hierarchy (Figure 3.3 of Section 3.5) not implied by $M\overline{O}_{c^e}$, $SM\overline{L}_{c^s}$ or $RM\overline{L}_{c^0}$ for $\mathcal{C}$.*

Proof. We need to show, that $\mathcal{F}$ does not achieve any of the lowest notions in the hierarchy that are not already implied by $M\overline{O}_{c^e}$, $SM\overline{L}_{c^s}$ or $RM\overline{L}_{c^0}$ for $\mathcal{C}$: $(SR)\overline{O}_{c^0}$, $(2S)\overline{O}_{c^0}$, $(2R)\overline{O}_{c^0}$, $SF\overline{L}-P_{c^0}$, $RF\overline{L}-P'_{c^0}$. This implies that also no stronger notions can be achieved, even without user corruption. Further, we show, that with differing behavior at corrupted receivers allowed it does not achieve $(SM)\overline{O}_{c^s}$, $(RM)\overline{L}_{c^s}$, $(RM)\overline{O}_{c^s}$, $M\overline{O}-|m|_{c^s}$, $RM\overline{L}-P'_{c^s}$, $(SR)\overline{L}_{c^e}$. This implies that no $M\overline{O}_{c^s}$ can be achieved. Obviously, as for corrupted senders all information about the communication is sent to the adversary, no notion can be achieved against differing behavior at corrupted senders allowed.

a)

	scenario 0	scenario 1
instance 0	A $\longrightarrow$ B C $\longrightarrow$ D	A $\longrightarrow$ D C $\longrightarrow$ B
instance 1	C $\longrightarrow$ D A $\longrightarrow$ B	C $\longrightarrow$ B A $\longrightarrow$ D

b)

	scenario 0	scenario 1
instance 0	A $\longrightarrow$ B	A $\longrightarrow$ D
instance 1	C $\longrightarrow$ D	C $\longrightarrow$ B

c)

	scenario 0	scenario 1	
instance 0	A $\longrightarrow$ B A $\longrightarrow$ B	A $\longrightarrow$ B C $\longrightarrow$ B	switch stage
instance 1	C $\longrightarrow$ B C $\longrightarrow$ B	C $\longrightarrow$ B A $\longrightarrow$ B	switch stage

Figure C.1: Similar to Figure C.1 from Section 3.3.2, depicting example inputs for notions: a) $(SR)\overline{L}$, b)$(SR)\overline{O}$, c)$(2S)\overline{O}$

$(SR)\overline{O}_{c^0}$. The attack works as follows: We use the communication of users A, B, C, D according to the definition of $(SR)\overline{O}$ (see Figure C.1 b).

Now, the ideal functionality $\mathcal{F}$ will output "Onion temp from S to X_1" with S being A or C. We will use *Deliver_Message* for *temp* and continue getting messages "Onion $\overline{temp}$ from X_1 to X_2" and using *Deliver_Message* for $\overline{temp}$ until we get a message "Onion temp from X_1 to R" with R being B or D. We guess the scenario that includes the linking between S and R.

$(2S)\overline{O}_{c^0}$ ($(2R)\overline{O}_{c^0}$ similarly). We use senders A and C for the two instances of the two scenarios according to Figure C.1 c). In this case, we do not need to use *Deliver_Message* even once; we just wait for the first messages the ideal functionality sends in *Process_Next_Step* "Onion temp from S_1 to X" and "Onion temp from S_2 to X", if those two senders are the same, we guess $g = 0$, otherwise $g = 1$.

$(SM)\overline{O}_{c^s}$. Analogous to $(SR)\overline{O}_{c^0}$, except that we exploit to pick a corrupted receiver and hence get the delivered message as output from the ideal functionality.

$(RM)\overline{L}_{c^s}$,$(RM)\overline{O}_{c^s}$, $M\overline{O}-|m|_{c^s}$, $RM\overline{L}-P'_{c^s}$. We are allowed to pick corrupted receivers, hence we do and get the receiver-message linking output from the ideal functionality in the "Onion from x with message m for r routed through ..."- message after we used *Deliver_Message* whenever possible.

$(SR)\overline{L}_{c^e}$. We choose a corrupted sender that sends to different receivers. Thus, we learn to which receiver the corrupted sender sends its message and hence learn the linking of the message and the receiver and win the game with certainty.

$SF\overline{L} - P_{c^0}(RF\overline{L} - P'_{c^0}$ similarly). We pick scenarios that differ in how often A sends, e.g. $b = 0$: A sends once, C k-times; $b = 1$: A sends k-times, C once. The ideal functionality will output the parts of the path of all communications. If A occurs more often in those parts of the path, we guess $g = 1$, if C occurs more often $g = 0$, otherwise we guess randomly.

Thus, we win if A (resp. C) is picked at most $k-2$ times as a random relay in $b = 0$ (resp. $b = 1$). This happens if it is not chosen as the common honest relay ($\frac{1}{\#honestrelays}$) and not chosen more often randomly as relay $((\frac{N-1}{\#P-1})^k + (\frac{N-1}{\#P-1})^{k-1} \cdot (\frac{1}{\#P-1}))$. Thus, we win with probablity of at least $1 - (\frac{1}{\#honestrelays} + (\frac{N-1}{\#P-1})^k + (\frac{N-1}{\#P-1})^{k-1} \cdot (\frac{1}{\#P-1}))$, which is a non negligible advantage if $\#P > N$ for an appropriately chosen k. □

Note 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.2. Proof of new Properties

C.2.1 No Replies

Our proof is in parts identical to the proof from [31]. We highlight the *identical parts* in blue. 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. We first describe the simulator $\mathcal{S}$. Then we show indistinguishability of the environment's view in the real and ideal world.

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, $\mathcal{S}$ sets-up the protocol: 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 $\mathcal{A}$.

There are two challenges for the simulator: First, it has to convincingly mimic the communications of honest senders for $\mathcal{A}$. As the environment initiates those communications in the ideal world, $\mathcal{S}$ has to use the limited information the ideal world gives about those communications to build onions in the simulated real world. Therefore, $\mathcal{S}$ needs to store the translation of the *temp* ID that was used in the ideal world with the onion $\mathcal{S}$ replaced it with. $\mathcal{S}$ stores those mappings on the r-list. Each entry $(onion_r, nextRelay, temp)$ represents the onion $onion_r$ that $\mathcal{S}$ expects to receive as honest party *nextRelay* from $\mathcal{A}$ (either from the link between honest parties or from an adversarial relay) and its corresponding *temp* ID. This *temp* ID is used to allow the onion to continue its path in $\mathcal{F}$ if the corresponding onion is sent to *nextRelay*. Secondly, $\mathcal{S}$ has to convincingly mimic the communications of adversarial senders in $\mathcal{F}$, such that $\mathcal{Z}$ does not notice a difference. In the case of an adversarial sender starting to communicate, $\mathcal{S}$ (as the honest relay) receives an onion from $\mathcal{A}$. $\mathcal{S}$ stores the processing of this onion together with the hop receiving the processing and all information on the O-list. As in $\mathcal{F}$ all information to communications with adversarial senders is output on every step of the path, $\mathcal{S}$ can easily map the correct onion to the communication once it occurs in the ideal functionality.

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 $(onion, nextRelay, information)$.

We now describe what the simulator does when it receives a message from the ideal functionality and then describe what it does when it receives a message from the adversary.

$\mathcal{S}$'s behavior on a message from $\mathcal{F}$. In case the received output belongs to an adversarial sender's communication[4]:

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 $(O_1, \ldots, O_n) \leftarrow$ 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. Processing is continued once O_{j+1} is sent by $\mathcal{A}$.

Case V: "Onion $temp$ from P_{o_i} routed through $(P_{o_{i+1}}, \ldots, P_{o_{j-1}})$ to P_{o_j}". In this case both P_{o_i} and P_{o_j} are honest, while the intermediate $(P_{o_{i+1}}, \ldots, P_{o_{j-1}})$ are adversarial. $\mathcal{S}$ 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 $(O_1, \ldots, O_n) \leftarrow$ 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 $(P_{o_{i+1}}, \ldots, P_{o_n})$". In this case, P_{o_i} is honest while everyone else is adversarial, including the recipient P_r. This means that some honest party sent a message to the dishonest party P_r. $\mathcal{S}$ 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 $(O_1, \ldots, O_n) \leftarrow$ 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}.

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

$\mathcal{S}$'s behavior on a message from $\mathcal{A}$ Let us now describe what the simulator $\mathcal{S}$ does upon receipt of a message from the adversary. Suppose the simulator $\mathcal{S}$, as real world honest party P_i, received an onion O from the adversary $\mathcal{A}$ as adversarial player P_a. Notice that this onion can be the protocol-conform processing of an onion from a communication of an honest sender, the non-protocol-conform processing of such an onion or the begin of a communication of an adversarial sender.

Case VII: $(O, P_i, temp)$ is on the r-list for some $temp$. Thus, 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 $\text{ProcOnion}(SK(P_i), O, P_i) = (O', P')$. We distinguish the case where a next hop exists and not.

(a) $P' = \perp$: P_{o_j} is the recipient and O' is a message or a fail symbol. This means that in the real-world protocol, this onion gets to real-life P_i, and P_i receives the message or outputs the fail report. 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.

This concludes the description of the simulator.

Indistinguishability

Let us now argue that the simulator actually works, i.e., that the distribution of the player's outputs in the real world and in the ideal world are the same. We proceed by a more or less standard hybrid argument. Consider the following set of hybrid machines:

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 $O_1, O_2, \dots, O_j$ from an honest sender P_0 to the next honest node P_j are replaced with $\bar{O}_1, \dots, \bar{O}_j$ where $\bar{O}_i = O'_{k+i}$ with $(O'_1, \dots, O'_n) \leftarrow \text{FormOnion}(m_{rdm}, \mathcal{P}_r dm, (PK)_{\mathcal{P}_{rdm}})$ 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 $(\bar{O}_j, P_j, \text{ProcOnion}(SK_{P_j}, O_j, P_j))$ 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 $\text{ProcOnion}(SK_{P_j}, O_j, P_j)$ is used as processing result of P_j. Otherwise, $\text{ProcOnion}(SK_{P_j}, \tilde{O}, P_j)$ is used.

$\mathcal{H}_0 \approx_I \mathcal{H}_1^1$. 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. We argue indistinguishability in the outputs to $\mathcal{A}$ as well:

$\mathcal{A}$ 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. Hence, if the case of more outputs cannot be distinguished, neither the case with less outputs can. 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[5]. For the other relays the attack decides on the keys as $\mathcal{A}$ (for corrupted) and the protocol (for honest) does. It receives $(\tilde{O}, \text{ProcOnion}(O_j))$ 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 $\text{ProcOnion}(O_j)$ 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 $\text{ProcOnion}(O_j)$. In the second case, $f(\tilde{O})$ is given to the oracle and the simulation is done for the returned $\text{ProcOnion}(f(\tilde{O}))$.

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.

$\boldsymbol{\mathcal{H}_1^1 \approx_I \mathcal{H}_1^*}$. Analogous above. Apply argumentation of indistinguishability ($\mathcal{H}_0 \approx_I \mathcal{H}_1^1$) for every replaced subpath.[6]

Hybrid $\mathcal{H}_2^1$. In this hybrid, for one communication (and all its replays) for which in the adversarial processing no modification occurred[7] 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 $(O'_1, \ldots, O'_n) \leftarrow \text{FormOnion}(m_{rdm}, \mathcal{P}_{rdm}, (PK)_{\mathcal{P}_{rdm}})$ 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 $(\bar{O}_{j'}, P_{j'}, \text{ProcOnion}(SK_{P_{j'}}, O_{j'}, P_{j'}))$ 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 $\text{ProcOnion}(SK_{P_{j'}}, O_{j'}, P_{j'})$ is used as processing result of $P_{j'}$. Otherwise, $\text{ProcOnion}(SK_{P_{j'}}, \tilde{O}, P_{j'})$ is used.

$\boldsymbol{\mathcal{H}_1^* \approx_I \mathcal{H}_2^1}$. $\mathcal{H}_2^1$ 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 $P_{j'}$.

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

[6]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.

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

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 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 in 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.

$\boldsymbol{\mathcal{H}_2^1 \approx_I \mathcal{H}_2^*}$. 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 $O_{j+1}, \dots, O_n$ from a communication of an honest sender, starting at the last honest node P_j to the corrupted receiver P_n are replaced with $\bar{O}_{j+1}, \dots, \bar{O}_n$. Thereby $\bar{O}_i = O'_{k+i}$ with $(O'_1, \dots, O'_{n'}) \leftarrow \text{FormOnion}(m, \mathcal{P}_{rdm}, (PK)_{\mathcal{P}_{rdm}})$ where m is the message of this communication[8], $\mathcal{P}_{rdm}$ a random path that includes the sequence from P_j to P_n starting at the k-th node.

$\boldsymbol{\mathcal{H}_2^* \approx_I \mathcal{H}_3^1}$. 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.

$\boldsymbol{\mathcal{H}_3^1 \approx_I \mathcal{H}_3^*}$. 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.

$\boldsymbol{\mathcal{H}_3^* \approx_I \mathcal{H}_4}$. 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

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

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.2.2 With Replies

Other OR Property Definitions

Definition 43 (Backward Layer-Unlinkability $LU^{\leftarrow}$)**.** *Backward Layer-Unlinkability is defined as:*

1. *The adversary receives the router names P_H, P_S and challenge public keys PK_S, PK_H, chosen by the challenger by letting $(PK_H, SK_H) \leftarrow G(1^\lambda, p, P_H)$ and $(PK_S, SK_S) \leftarrow G(1^\lambda, p, P_S)$.*

2. *Oracle access: The adversary may submit any number of* Proc *and* Reply *requests for P_H or P_S to the challenger. For any* Proc(P_H, O)*, the challenger checks whether η is on the η^H-list. If not, it sends the output of* ProcOnion(SK_H, O, P_H)*, stores η on the η^H-list and O on the O^H-list. For any* Reply(P_H, O, m) *the challenger checks if O is on the O^H- list and if so, the challenger sends* ReplyOnion(m, O, P_H, SK_H) *to the adversary. (Similar for requests on P_S with the η^S-list).*

3. *The adversary submits*

   - *message m,*

   - *a position $j^{\leftarrow}$ with $0 \leq j^{\leftarrow} \leq n^{\leftarrow} + 1$,*

   - *a path $\mathcal{P}^{\rightarrow} = (P_1, \dots, P_j, \dots, P_{n+1})$, where $P_{n+1} = P_H$, if $j^{\leftarrow} = 0$,*

   - *a path $\mathcal{P}^{\leftarrow} = (P_1^{\leftarrow}, \dots, P_{j^{\leftarrow}}^{\leftarrow}, \dots, P_{n^{\leftarrow}+1}^{\leftarrow} = P_S)$ with the honest node P_H at backward position $j^{\leftarrow}$, if $1 \leq j^{\leftarrow} \leq n^{\leftarrow} + 1$, and the second honest node P_S at position $n^{\leftarrow} + 1$*

   - *and public keys for all nodes PK_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 chosen paths are acyclic, the router names are valid and that the same key is chosen if the router names are equal, and if so, sets $PK_{j^{\leftarrow}}^{\leftarrow} = PK_H$ (resp.*

PK_{n+1} if $j^{\leftarrow} = 0$), $PK^{\leftarrow}_{n^{\leftarrow}+1} = PK_S$ and sets bit b at random.

5. *The challenger creates the onion with the adversary's input choice and honestly chosen randomness $\mathcal{R}$:*

$$O_1 \leftarrow \text{FormOnion}(1, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}})$$

and sends O_1 to the adversary.

6. *The adversary gets oracle access as in step 2) except if:*

Exception 1) The request is ...

- *for $j^{\leftarrow} > 0$: Proc(P_H, O) with* $\text{RecognizeOnion}((n+1) + j^{\leftarrow}, O, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}) = True$*, η is not on the η^H-list and* $\text{ProcOnion}(SK_H, O, P_H) \neq \perp$*: stores η on the η^H and O on the O^H-list and ...*

- *for $j^{\leftarrow} = 0$: Reply$(P_H, O, m^{\leftarrow})$ with* $\text{RecognizeOnion}((n+1), O, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}) = True$*, O is on the O^H- list and no onion with this η has been replied to before and* $\text{ReplyOnion}(m^{\leftarrow}, O, P_H, SK_H) \neq \perp$*:*

.. then: The challenger picks the rest of the return path $\bar{\mathcal{P}}^{\rightarrow} = (P^{\leftarrow}_{j^{\leftarrow}+1}, \dots, P^{\leftarrow}_{n^{\leftarrow}+1})$, an empty backward path $\bar{\mathcal{P}}^{\leftarrow} = ()$, and a random message $\bar{m}$, another honestly chosen randomness $\bar{\mathcal{R}}$, and generates:

$$\bar{O}_1 \leftarrow \text{FormOnion}(1, \bar{\mathcal{R}}, \bar{m}, \bar{\mathcal{P}}^{\rightarrow}, \bar{\mathcal{P}}^{\leftarrow}, (PK)_{\bar{\mathcal{P}}^{\rightarrow}}, (PK)_{\bar{\mathcal{P}}^{\leftarrow}})$$

- *If $b = 0$, the challenger calculates* $(O_{j^{\leftarrow}+1}, P^{\leftarrow}_{j^{\leftarrow}+1}) = \text{ProcOnion}(SK_H, O, P^{\leftarrow}_{j^{\leftarrow}})$ *(for $j^{\leftarrow} > 0$) resp.* $(O_{j^{\leftarrow}+1}, P^{\leftarrow}_{j^{\leftarrow}+1}) = \text{ReplyOnion}(m^{\leftarrow}, O, P^{\leftarrow}_{j^{\leftarrow}}, SK_H)$ *(for $j^{\leftarrow} = 0$) and gives $O_{j^{\leftarrow}+1}$ for $P^{\leftarrow}_{j^{\leftarrow}+1}$ to the adversary.*

- *Otherwise, the challenger gives $\bar{O}_1$ for $P^{\leftarrow}_{j^{\leftarrow}+1}$ to the adversary.*

Exception 2) Proc(P_S, O) with O being the challenge onion as processed for the final receiver on the backward path, i.e.:

- *for $b = 0$:* $\text{RecognizeOnion}((n+1) + (n^{\leftarrow}+1), O, \mathcal{R}) = True$

- *for $b = 1$:* $\text{RecognizeOnion}((n^{\leftarrow}+1) - j^{\leftarrow}, O, \bar{\mathcal{R}}, \bar{m}, \bar{\mathcal{P}}^{\rightarrow}, \bar{\mathcal{P}}^{\leftarrow}, (PK)_{\bar{\mathcal{P}}^{\rightarrow}}, (PK)_{\bar{\mathcal{P}}^{\leftarrow}}) = True$

.. then the challenger outputs nothing.

7. *The adversary produces guess b' .*

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

Definition 44 (Repliable Tail-Indistinguishability $TI^{\leftrightarrow}$)**.** *Repliable Tail-Indistinguishability is defined as:*

1. *The adversary receives the router names $P_H, P^{\leftarrow}_H, P_S$ and challenge public keys $PK_S, PK_H, PK^{\leftarrow}_H$, chosen by the challenger by letting $(PK_H, SK_H) \leftarrow G(1^{\lambda}, p, P_H)$, $(PK^{\leftarrow}_H, SK^{\leftarrow}_H) \leftarrow G(1^{\lambda}, p, P^{\leftarrow}_H)$, $(PK_S, SK_S) \leftarrow G(1^{\lambda}, p, P_S)$.*

2. *Oracle access: The adversary may submit any number of Proc and Reply requests for $P_H, P^{\leftarrow}_H$ or P_S to the challenger. For any Proc(P_H, O), the challenger checks whether η is on the η^H-list. If not, it sends the output of* $\text{ProcOnion}(SK_H, O, P_H)$*, stores η on the η^H-list and O on the O^H-list. For any Reply(P_H, O, m) the challenger checks if O is on the O^H- list and if so, the challenger sends* $\text{ReplyOnion}(m, O, P_H, SK_H)$ *to the adversary. (Similar for requests on $P^{\leftarrow}_H, P_S$).*

3. *The adversary submits a message m, a path $\mathcal{P}^{\rightarrow} = (P_1, \dots, P_j, \dots, P_{n+1})$ with the honest node P_H or $P_H^{\leftarrow}$ at position j, $1 \leq j < n+1$, a path $\mathcal{P}^{\leftarrow} = (P_1^{\leftarrow}, \dots, P_{n^{\leftarrow}+1}^{\leftarrow})$ with the honest node $P_H^{\leftarrow}$ at position $1 \leq j^{\leftarrow} \leq n^{\leftarrow}+1$ and public keys for all nodes PK_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 given paths are acyclic, the router names are valid and that the same key is chosen if the router names are equal, and if so, sets $PK_j = PK_H$ (or $PK_j = PK_H^{\leftarrow}$, if the adversary chose $P_H^{\leftarrow}$ at this position as well) , $PK_{j^{\leftarrow}}^{\leftarrow} = PK_H^{\leftarrow}$,$PK_{n^{\leftarrow}+1}^{\leftarrow} = PK_S$ and sets bit b at random.*

5. *The challenger creates the onion with the adversary's input choice and honestly chosen randomness $\mathcal{R}$:*

$$O_{j+1} \leftarrow \text{FormOnion}(j+1, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}})$$

and a replacement onion with the path from the honest relay P_H to the corrupted receiver $\bar{\mathcal{P}}^{\rightarrow} = (P_{j+1}, \dots, P_{n+1})$ and the backward path from the corrupted receiver starting at position 0 ending at $j^{\leftarrow}$: $\bar{\mathcal{P}}^{\leftarrow} = (P_1^{\leftarrow}, \dots, P_{j^{\leftarrow}}^{\leftarrow})$; and another honestly chosen randomness $\bar{\mathcal{R}}$:

$$\bar{O}_1 \leftarrow \text{FormOnion}(1, \bar{\mathcal{R}}, m, \bar{\mathcal{P}}^{\rightarrow}, \bar{\mathcal{P}}^{\leftarrow}, (PK)_{\bar{\mathcal{P}}^{\rightarrow}}, (PK)_{\bar{\mathcal{P}}^{\leftarrow}})$$

6. *If $b = 0$: The challenger sends O_{j+1} to the adversary.*
   Otherwise: The challenger sends $\bar{O}_1$ to the adversary.

7. *Oracle access: the challenger processes all requests as in step 2) except if...*

   ... Proc$(P_H^{\leftarrow}, O)$ *with O being the challenge onion as processed for the honest relay on the backward path, i.e.:*

   - *for* $b = 0 : \text{RecognizeOnion}((n+1) + j^{\leftarrow}, O, \mathcal{R}) = True$ *or*
   - *for* $b = 1 : \text{RecognizeOnion}((n-j) + j^{\leftarrow}, O, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}) = True$

   .. then the challenger outputs nothing.

8. *The adversary produces guess b'.*

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

Proof of UC-realization

The argumentation extends the one from Appendix C.2.1 for the replies.

Informally: For corrupted sender (= backward receiver), all information about the communication are leaked in the ideal functionality. Thus, no protection is needed.

For honest senders (= backward receivers), we want to ensure that only the subpaths between honest relays and (if the receiver is corrupted) the messages can be learned by the adversary. Therefore, we start by replacing the onion layers on the first part of the path, i.e. from the honest sender to the first honest relay on the forward path, with random ones that take the same path. Due to $LU^{\rightarrow}$, we know that the adversary cannot notice the difference. We continue, one onion and subpath at the time, until all subpaths between honest relays on the forward path are replaced.

Next, we replace the last part of the backward path, i.e. from the last honest relay on the backward path to the honest sender (= backward path receiver). Due to $LU^{\leftarrow}$, we know that the adversary cannot notice the difference. We continue, one onion and subpath at the time, until all subpaths between honest relays on the backward path are replaced.

If the receiver is honest, the steps above already replaced everything, as then the receiver is an honest relay. If the receiver is corrupted, we still need to replace the subpath between the last honest relay on the forward and the first honest relay on the backward path. We can do this without the adversary noticing any change due to $TI^{\leftrightarrow}$. Thus, we replaced the onion layers on all subpaths.

Formally: We assume that the public keys are already distributed and define any secure, repliable OR scheme to fulfill our properties:
Definition 45. *A secure repliable OR scheme is a quadruple of polynomial-time algorithms* (G, $FormOnion, ProcOnion, ReplyOnion$) *(Section 6.3.2) that achieves Onion-Correctness (Def. 22), Repliable Tail-Indistinguishability (Def. 44), Forward Layer-Unlinkability (Def. 23) and Backward Layer-Unlinkability (Def. 43).*

Similarly to [31] and Chapter 5, we say that a OR protocol is build from the OR scheme with an additional ideal functionality for the assumed key distribution $\mathcal{F}_{RKR}$.
Definition 46. *OR protocol* Π *is a secure repliable OR protocol (in the* $\mathcal{F}_{RKR}$*-hybrid model), iff it is based on a secure OR scheme* $(G, \text{FormOnion}, \text{ProcOnion}, \text{ReplyOnion})$ *and works as follows:*

Setup: *Each node* P_i *generates a key pair* $(SK_i, PK_i) \leftarrow G(1^\lambda)$ *and publishes* PK_i *by using* $\mathcal{F}_{RKR}$.

Sending a Message: *If* P_S *wants to send* $m \in \mathcal{M}$ *to* P_R *over path* $P_1, \ldots, P_n$ *with* $n < N$ *and wants to allow a reply over the path* $P_1^{\leftarrow}, \ldots, P_{n^{\leftarrow}}^{\leftarrow}$ *with* $n^{\leftarrow} < N$ *and* $P_{n^{\leftarrow}}^{\leftarrow} = P_S$*, he chooses a randomness* $\mathcal{R}$ *and sends the following* O_1 *to* P_1.

$$\begin{aligned} O_1 \leftarrow &\text{FormOnion}(1, \mathcal{R}, m, (P_1, \ldots, P_n, P_R), (P_1^{\leftarrow}, \ldots, P_{n^{\leftarrow}}^{\leftarrow}), \\ &(PK_1, \ldots, PK_n, PK_R), (PK_1^{\leftarrow}, \ldots, PK_{n^{\leftarrow}}^{\leftarrow})) \end{aligned}$$

Replying an Onion: *If* P_R *wants to reply to an onion* O *with message* $m^{\leftarrow}$*, he sends* $O_1^{\leftarrow}$ *to* $P_1^{\leftarrow}$ *which are calculated as*

$$(O_1^{\leftarrow}, P_1^{\leftarrow}) \leftarrow \text{ReplyOnion}(m^{\leftarrow}, O, P_R, SK_R).$$

Processing an Onion: *If* P_i *received* O_i*, he calculates:*

$$(O_j, P_j) \leftarrow \text{ProcOnion}(SK_i, O_i, P_i)$$

If $P_j = \perp$*,* P_i *outputs "Received* $(m, Reply) = O_j$*" 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.

We now show that our properties are sufficient for the ideal functionality:
Theorem 14. *A secure repliable onion routing protocol following Definition 46 UC-realizes* $\mathcal{F}$ *in the* $(\mathcal{F}_{RKR})$*-hybrid model.*

Therefore, we describe a simulator that translates any attack on the secure, repliable OR protocol to an attack in the ideal functionality.

Simulator Overview

The simulator uses the knowledge of honest keys to process adversarial onions (FormOnion called by an adversarial sender or modified at the adversarial relay) just as the protocol does. For honest onions (FormOnion called by an honest sender) our simulator uses the information it gets from the ideal functionality to build the random replacement onions for each subpath. This information are the part of the path (and if the receiver is adversarial, the message and repliability). The correct relaying of honest onions is recognized by the simulator with *RecognizeOnion*. With the help of our security properties, we can show that the adversary cannot notice the change. We give an overview over standard hybrid argument in Table C.1. The full proof that can be found in [98] is rather extensive and hence omitted from this thesis.

C.3. Additional Results without Replies

C.3.1 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.

We illustrate this idea in Fig. C.2.

Message in Π_{broken}	Encoding of taken path $P_0 \| \mathcal{P}$

Figure C.2: Illustration of the message extension used as message in Π

More formally, our extended protocol $\Pi_{broken1}$ using FormOnion$_{broken1}$ and ProcOnion$_{broken1}$ is created from the "secure" onion routing protocol Π from [31]. 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 FormOnion$_\Pi$ and ProcOnion$_\Pi$.

Sender [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. 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. FormOnion$_{broken1}$ runs the original algorithm FormOnion$_\Pi$ with the inputs chosen by the sender except that the message is replaced with m'.

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

Receiver [ProcOnion$_{broken1}$]. The receiver getting O_{n+1} executes 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.

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.

Table C.1: Overview Proof: Properties imply Ideal Functionality

Hybrid	**Description**	**Reduction**
$\mathcal{H}_0$	Machine using the real world protocol to interact with the real world adversary $\mathcal{A}$ and the environment	
$\mathcal{H}_1 = \mathcal{H}_1^{<2}$	As $\mathcal{H}_0$ but for one forward communication of an honest sender: The onion layers between this sender and the next honest node (relay or receiver) are replaced by the layers of a newly formed onion taking this part of this path but carrying a random message for the next honest node.	$LU^{\rightarrow}$
$\mathcal{H}_1^{<x}$	As $\mathcal{H}_1^{<x-1}$ but for one forward communication of an honest sender, where the onion layers between the first two honest nodes are not yet replaced: Replace as in $\mathcal{H}_1$	$(LU^{\rightarrow})$
$\mathcal{H}_2 = \mathcal{H}_2^{<2}$	As $\mathcal{H}_1^*$ (=first part for all honest forwards communications replaced) but for one forward communication of an honest sender where no modification happened: The onion layers between the next two honest nodes (relay or receiver) is replaced as in $\mathcal{H}_1$	$LU^{\rightarrow}$
$\mathcal{H}_2^{<x}$	As $\mathcal{H}_2^{<x-1}$ but for one forward communication of an honest sender where no modification happened and these layers are not yet replaced: Replace as in $\mathcal{H}_2$	$(LU^{\rightarrow})$
$\mathcal{H}_1^{\leftarrow} = \mathcal{H}_1^{<2\leftarrow}$	As $\mathcal{H}_2^*$ (=all are replaced) but for one backward communication of an honest sender: The onion layers between the last honest node (relay or forward receiver) are replaced by the layers of a newly formed onion taking this part of this path but carrying a random message for the honest backward receiver (=forward sender).	$LU^{\leftarrow}$
$\mathcal{H}_1^{<x\leftarrow}$	As $\mathcal{H}_1^{<x-1\leftarrow}$ but for one backward communication of an honest sender, where the onion layers between the last honest node and (backward) receiver are not yet replaced: Replace as in $\mathcal{H}_1^{\leftarrow}$	$(LU^{\leftarrow})$
$\mathcal{H}_2^{\leftarrow} = \mathcal{H}_2^{<2\leftarrow}$	As $\mathcal{H}_1^{*\leftarrow}$(=all are replaced) but for one backward communication of an honest sender where no modification happened: The onion layers between the (next) last two honest nodes (relay or forward receiver) are replaced as in $\mathcal{H}_1^{\leftarrow}$	$LU^{\leftarrow}$
$\mathcal{H}_2^{<x\leftarrow}$	As $\mathcal{H}_2^{<x-1\leftarrow}$ but for one forward communication of an honest sender where no modification happened and these layers are not yet replaced: Replace as in $\mathcal{H}_2^{\leftarrow}$	$(LU^{\leftarrow})$
$\mathcal{H}_3 = \mathcal{H}_3^{<2}$	As $\mathcal{H}_2^{*\leftarrow}$(=all are replaced) but for one forward communication of an honest sender where no modification happened but no other honest relay exists(i.e. receiver is corrupt): The onion layers between the last honest node on the forward path and the receiver are replaced with the ones generated by a newly formed onion for this part of the path, carrying the same message	$TI^{\leftrightarrow}$
$\mathcal{H}_3^{<x}$	As $\mathcal{H}_3^{<x-1}$ but for one forward communication of an honest sender where no modification happened and these layers are not yet replaced: Replace as in $\mathcal{H}_3$	$(TI^{\leftrightarrow})$

C.3.2 Insecure Protocol 2

The main idea for this scheme is to take the secure onion routing scheme of [31] and append the same identifier ID and another extension to all onion layers of an onion. Our new onion layer is $O_i \| ID \| ext_i$, where O_i denotes the onion layer of the original protocol and $\|$ denotes concatenation. The ID makes the onion easily traceable, as it stays the same while processing the onion at a relay. To assure that the properties are still achieved, the additional extension ext_i has special characteristics, like to prohibit modification. In this subsection, we first explain how ext_i is built and processed. Then we describe and analyze the scheme based on ext_i's characteristics.

Extension. As mentioned, to achieve the properties although we attach an ID, we need an extension to protect the ID from modification. More precisely, the extension needs to fulfill the following characteristics: It cannot leak information about the inputs to FormOnion as otherwise Onion-Security breaks. Further, we need to prohibit modification of the appended ID and extension as otherwise the oracle in Onion-Security can be used with a modified ID or extension to learn the next hop and break Onion-Security. Third, we require the extension to be of fixed length to easily determine where it starts. Thus, we need an extension ext_i with the following characteristics to be created by the sender that knows all O_i's (created by a protocol with Onion-Security) and picks the ID randomly:

1. ext_i does not leak more information about the onion that it is attached to than the onion already leaks.

2. Except with negligible probability any change of $ID \| ext_i$ is detected in ProcOnion of P_i and the processing aborted.

3. ext_i has a known fixed length.

Scheme Description. We describe an extension with the above criteria in Appendix C.3.3. With it, we can create the extended protocol $\Pi_{broken2}$ using $\text{FormOnion}_{broken2}$ and $\text{ProcOnion}_{broken2}$ from Π:

Let Π be the onion routing protocol from [31] that 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 FormOnion_Π and ProcOnion_Π. Let O_i be the i-th onion layer of an onion of Π. In our new scheme the i-th onion layer is $O_i \| ID \| ext_i$.

Sender [$\text{FormOnion}_{broken2}$]. The sender runs the original algorithm FormOnion_Π. Additionally, as part of the new $\text{FormOnion}_{broken2}$ a random number ID of fixed length is picked. The sender creates an extension ext_i for all layers i and appends the ID and extension to the onion layers to generate the output of $\text{FormOnion}_{broken2}$. The resulting onion layer $O'_1 = O_1 \| ID \| ext_1$ is sent to P_1.

Intermediate Router [$\text{ProcOnion}_{broken2}$]. Any intermediate router receiving $O'_i = O_i \| ID \| ext_i$ uses the fixed length of ID and ext_i (characteristic 3) to split the parts of the onion layer. This way it retrieves the original onion layer O_i. Then it runs ProcOnion_Π on O_i to create O_{i+1}. Afterwards it processes ext_i (aborts if the extension was modified before) to generate ext_{i+1}. Finally, it sends the layer $O'_{i+1} = O_{i+1} \| ID \| ext_{i+1}$ to P_{i+1}.

Receiver [Identical to first part of $\text{ProcOnion}_{broken2}$]. Getting $O'_{n+1} = O_{n+1} \| ID \| ext_{n+1}$, the receiver splits the parts of the onion layer. Thereby it retrieves the original layer O_{n+1} and executes ProcOnion_Π on it.

Analysis regarding properties. The properties follow from the corresponding properties of the original protocol. As we only add and remove $ID\|ext$, the same path is taken and the onion layers O_i are calculated as before. Hence, *Correctness* and *Integrity* hold. The first part of the onion is not changed in our modification of the protocol and hence, *re-wrapping* this part is as difficult as before. Only *Onion-Security* remains. As Π has Onion-Security, it can only be broken in $\Pi_{broken2}$ by learning information from $ID\|ext$ (with or without modification). The ID is chosen randomly and independent from FormOnion's input and security parameters. Hence, it does not depend on any information the adversary has to distinguish in the Onion-Security game, i.e. the path and message. By characteristic 1 the extension ext_i is not leaking more information than O_i was leaking before, which is not enough to win the Onion-Security game as Π has Onion-Security. Only learning by modifying remains. As any modification of $ID\|ext$ is detected (characteristic 2), the adversary cannot create an onion layer with modified $ID\|ext$ that would be processed by the oracle. Hence, adding the extension does not introduce any advantage in breaking Onion-Security.

Practical Insecurity. ID is never changed and hence the onion layers can be linked based on it. If the adversary observes the link from the sender and corrupts the receiver, the message and sender can be linked. If the adversary observes the link from the sender and the link to the receiver, the sender and receiver can be linked. Instead of observing the link, it is also sufficient to observe as the relay connected to this link[9]. In this case, the relay learns the ID as part of the onion and hence additional hop-to-hop encryption is not sufficient from a practical standpoint either.

C.3.3 Extension for Onion Security

The basic idea of our extension is to add one MAC for every router to protect $ID\|ext_i$ from modification. Further, similar to the scheme of [31] we use deterministic padding to ensure ext_i has a fixed length.

To build this extension, we require a number of primitives:

- $Enc_{asym} : \mathcal{K}_{Pub} \times \mathcal{M}_{asym} \to \mathcal{C}_{asym}$: a non-malleable CCA secure asymmetric encryption function for which public keys are shared in advance
- $Dec_{asym} : \mathcal{K}_{Priv} \times \mathcal{C}_{asym} \to \mathcal{M}_{asym}$: decryption function to Enc_{asym}
- $MAC : \mathcal{K}_{sym} \times \mathcal{M}_{MAC} \mapsto \mathcal{T}$: secure symmetric MAC scheme; modelled as a pseudo-random function (PRF)[10]
- $Ver : \mathcal{K}_{sym} \times \mathcal{T} \times \mathcal{M}_{MAC} \mapsto \{0,1\}$: verification algorithm to MAC
- $PRNG : \mathcal{K}_{sym} \times \mathbb{N} \to \mathcal{C}_{asym}$: secure PRNG
- $embed : \mathcal{K}_{sym} \times \mathcal{T} \to \mathcal{M}_{asym}$: bijective map
- $extract : \mathcal{M}_{asym} \to \mathcal{K}_{sym} \times \mathcal{T}$: inverse operation $extract = embed^{-1}$

Recall, that we do assume that a PKI is in place and that the maximum path length is N.

Forming Extensions: We treat ext_i as a concatenated vector of N blocks (see Fig. C.3), each being an element of the asymmetric ciphertextspace $c \in \mathcal{C}_{asym}$. We split this vector in the blocks that contain MACs tag_i and the padding blocks that contain only random numbers rdm_i to guarantee a constant length $N \cdot |c|$, i.e. for all i: $|(tag_i\|rdm_i)| = N \cdot |c|$.

First the random numbers are chosen. The last block is a pseudo-random number chosen by the previous relay (or the sender if no previous relay exists and the path is shorter than N). The other blocks of rdm_i are the result of the same choice at earlier nodes. We use pseudo-random numbers

[9]We assume the sender is not spoofing its address.

[10]This holds for constructions like HMAC using a PRF as compression function (like SHA256).

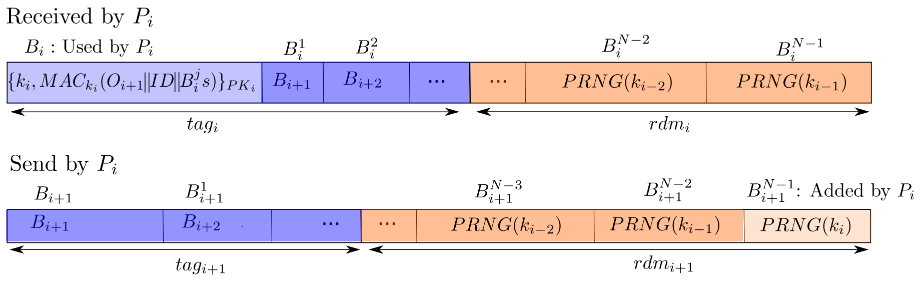


Figure C.3: ext_i (resp. ext_{i+1}). The current MAC block B_i used by P_i is depicted in light blue. Dark blue blocks are MAC blocks for upcoming routers, orange blocks depict the deterministic padding. $\{X\}_k$ is used to denote asymmetric encryption under key k. PK_i is the public key of P_i. B_i^j is short for all concatenated blocks that follow in the extension, i.e. $j \in \{1, \dots, N-1\}$.

to be able to calculate them in advance at the sender. Then the MAC blocks tag_i are calculated. Each such block B_i of tag_i is encrypted with the corresponding router's public key and has an ephemeral symmetric key k_i and a MAC t_i embedded. The MAC t_i authenticates the other $N-1$ received blocks of ext_i, as well as the chosen ID and next onion layer O_{i+1}.

Processing Extensions: The extension is checked for the length and processing is aborted if the extension length does not match $N \cdot |c|$. Otherwise, the first block of the extension is removed, decrypted with the secret key, the included ephemeral key k_i is extracted and the included MAC is verified. If the validation fails, the onion is discarded. Otherwise, a new block r_i is added by generating a pseudo-random number based on k_i.

Analysis for Characteristics As the extension only consists of encrypted MACs and pseudo-random numbers it does not leak more information about the onion than the adversary had before (characteristic 1). If $ID\|ext_i$ is changed, it might be modified or completely replaced. If it is modified either the MAC itself or the input to the MAC is modified. Thus, except with negligible probability the verification of the MAC fails and the onion is discarded. To replace it completely the adversary would need to know the next onion layer after the honest node O_{i+1}, which she cannot as the original scheme has Onion-Security. Thus, a changed $ID\|ext_i$ is discarded (characteristic 2). The attached padding blocks assure a fixed length of $N \cdot |c|$ (characteristic 3).

C.3.4 Sphinx

Adapted Sphinx

The original Sphinx protocol was adapted in [14] 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 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 [14].

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 5.2.1. 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.

Proof of adapted Sphinx

The proof for Onion-Correctness is analogous to the one in [52]. 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 [52] 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 [52] still holds.

LU : Recall that LU allows the adversary to decide the inputs to FormOnion and either returns the resulting onion O_1 of this 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 (ProcOnion(O_j)). Furthermore, it allows oracle use before and after this decision.

No dependencies between 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 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 $O_{+j} :=$ 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 ($\text{ProcOnion}(O_j) = (O_{j+1}, P_{j+1})$) or the processing from a random onion that matches the end of the path and message of the adversary's choice ($\text{ProcOnion}(\bar{O}_k) = (\bar{O}_{k+1}, P_{j+1})$). 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.

C.3.5 Correctness Practical Considerations

The false-positive probability of a Bloom filter depends on its configuration; that is its size m, the number of hash functions k and the number of already stored inputs n. The false-positive probability $p_{k,n,m}$ is:

$$p_{k,n,m} = \frac{1}{m^{k(n+1)}} \sum_{i=1}^{m} i^k i! \binom{m}{i} \left\{ {kn \atop i} \right\}$$

As a protocol is used, the duplicate store is growing, which means that for a Bloom filter the number of stored elements n grows. It follows that the false-positive rate increases and thus the probability for a correctness failure.

We thus extend Onion-Correctness to δ-Onion-Correctness such that the probability for a correctness failure is at most δ. Practically, this can be achived by computing the number of maximum elements that can be stored for a given Bloom filter configuration m, k such that $p_{k,n,m} \leq \delta$. Once the maximum number is achieved, the system would need to switch keys to restart duplicate detection using an empty Bloom filter.

Definition 47. *(δ-OnionCorrectness) [as in Definition 15]... the following is true:*

1. *correct path:* $Pr[\mathcal{P}(O_1, P_1) = (P_1, \ldots, P_{n+1})] \geq 1 - \delta$,
2. *correct layering:* $Pr[\mathcal{L}(O_1, P_1) = (O_1, \ldots, O_{n+1})] \geq 1 - \delta$,
3. *correct decryption:* $Pr[(m, \perp) = \text{ProcOnion}(SK(P_{n+1}), O_{n+1}, P_{n+1})] \geq 1 - \delta$.

C.4. Additional Results with Replies

C.4.1 Definition of Building Blocks

For our construction we make use of SUF-CMA secure MACs, PRP-CPA secure symmetric encryption, rerandomizable CPA secure public-key encryption, CCA2 secure public-key encryption, RCCA secure UE scheme with plaintext integrity, and simulation-sound SNARGs which are all defined below.

Definition 48. *A* message authentication code (MAC) *scheme consists of three PPT algorithms* $(\mathsf{Gen}, \mathit{MAC}, \mathit{Ver})$ *with the following syntax:*

Key generation. $\mathsf{Gen}(1^\lambda)$ *outputs a key* k.

MAC generation. $\mathit{MAC}(k, M)$ *computes a tag* γ *for a message* $M \in \{0,1\}^*$ *under key* k.

MAC Verification. $\mathit{Ver}(k, M, \gamma)$ *outputs a bit on input of a key* k*, a message* m*, and a tag* γ*.*

We require correctness in the sense that for all $\lambda \in \mathbb{N}$, all $k \leftarrow \mathsf{Gen}(1^\lambda)$, all $M \in \{0,1\}^*$, it holds that $\mathit{Ver}(k, M, \mathit{MAC}(k, M)) = 1$.

Finally, we say that the MAC scheme is SUF-CMA secure if for all PPT adversaries $\mathcal{A}$ it holds that the success probability defined by

$$\Pr\left[\begin{array}{c} \mathit{Ver}(k, M^*, \gamma^*) = 1 \\ \wedge \\ (M^*, \gamma^*) \notin \{(M_1, \gamma_1), \ldots, (M_q, \gamma_q)\} \wedge \end{array} \middle| \begin{array}{c} k \leftarrow \mathsf{Gen}(1^\lambda) \\ (M^*, \gamma^*) \leftarrow \mathcal{A}^{\mathit{MAC}(k,\cdot)}(1^\lambda) \end{array}\right]$$

is negligible in λ, where $\mathit{MAC}(k, \cdot)$ is an oracle that, on input M, returns $\mathit{MAC}(k, M)$, $M_1, \ldots, M_q$ denotes the messages queried by $\mathcal{A}$ to its oracle, and $\gamma_1, \ldots, \gamma_q$ the respective replies.

Furthermore, we will make use of PRP-CCA secure symmetric encryption for constructing onion headers which is defined as follows:

Definition 49. *A* symmetric encryption scheme *consists of three polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

Key generation. $\mathsf{Gen}(1^\lambda)$ *is a probabilistic algorithm which outputs a key k.*

Encryption. $\mathsf{Enc}(k, M)$ *is a deterministic algorithm which takes a key k and a plaintext message $M \in X \subset \{0,1\}^*$ and outputs a ciphertext C.*

Decryption. $\mathsf{Dec}(k, C)$ *is a deterministic algorithm takes a key k and a ciphertext C and outputs a plaintext message m.*

We require correctness in the sense that for all $\lambda \in \mathbb{N}$, all $k \leftarrow \mathsf{Gen}(1^\lambda)$, all $M \in X$, it holds that $\mathsf{Dec}(k, \mathsf{Enc}(k, M)) = M$.

We restrict to encryption schemes defining a permutation, i.e., for all $\lambda \in \mathbb{N}$ and $k \leftarrow \mathsf{Gen}(1^\lambda)$ the function $\mathsf{Enc}(k, \cdot)$ is a permutation on X. We call such an encryption scheme PRP-CCA secure if for every PPT $\mathcal{A}$, the advantage

$$\Pr\left[b' = b \middle| \begin{array}{c} k \leftarrow \mathsf{Gen}(1^\lambda) \\ P \leftarrow \mathit{Perm}(X) \\ b \leftarrow \{0,1\} \\ (F, F^{-1}) := \begin{cases} (\mathsf{Enc}(k, \cdot), \mathsf{Dec}(k, \cdot)), & b = 0 \\ (P, P^{-1}), & b = 1 \end{cases} \\ b' \leftarrow \mathcal{A}^{F(\cdot), F^{-1}(\cdot)}(1^\lambda) \end{array} \right] - \frac{1}{2}$$

is negligible in λ, where P is a uniformly chosen permutation on X and P^{-1} is its inverse. Note that if an encryption scheme is PRP-CCA secure then the above property also holds if we swap Enc *and* Dec *(i.e.,* Dec *is also a strong pseudo-random permutation on X).*

To provide payload integrity and unlinkability we make use of Updatable Encryption in our first scheme and therefore recapitulate the definitions from [89] below.

Definition 50 (Updatable Encryption [89])**.** *An* ***updatable encryption scheme*** UE *is a tuple* $(\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{GenTok}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReEnc})$ *of PPT algorithms defined as:*

$\mathsf{UE.GenSP}(pp)$ *is given the public parameters and returns some system parameters sp. We treat sp as implicit input to all other algorithms.*

$\mathsf{UE.GenKey}(sp)$ *is the key generation algorithm which on input of the system parameters outputs a key $k \in \mathcal{K}_{sp}$.*

$\mathsf{UE.GenTok}(k_e, k_{e+1})$ *is given two keys k_e and k_{e+1} and outputs some update token Δ_e.*

$\mathsf{UE.Enc}(k_e, M)$ *is given a key k_e and a message $M \in \mathcal{M}_{sp}$ and outputs some ciphertext $C_e \in \mathcal{C}_{sp}$ (or $\perp$ in case $M = \perp$).*

$\mathsf{UE.Dec}(k_e, C_e)$ *is given a key* k_e *and a ciphertext* C_e *and outputs some message* $m \in \mathcal{M}_{sp}$ *or* $\perp$.

$\mathsf{UE.ReEnc}(\Delta_e, C_e)$ *is given an update token* Δ_e *and a ciphertext* C_e *and returns an updated ciphertext* C_{e+1} *or* $\perp$.

Given UE, *we call* $\mathsf{SKE} = (\mathsf{GenSP}, \mathsf{GenKey}, \mathsf{Enc}, \mathsf{Dec})$ *the* ***underlying (standard) encryption scheme****.* UE *is called* ***correct*** *if* SKE *is correct and it holds that* $\forall sp \leftarrow \mathsf{GenSP}(pp), \forall k^{\text{old}}, k^{\text{new}} \leftarrow \mathsf{GenKey}(sp), \forall \Delta \leftarrow \mathsf{GenTok}(k^{\text{old}}, k^{\text{new}}), \forall C \in \mathcal{C}\colon \mathsf{Dec}(k^{\text{new}}, \mathsf{ReEnc}(\Delta, C)) = \mathsf{Dec}(k^{\text{old}}, C)$.

Definition 51 (UP-IND-RCCA [89])**.** UE *is called UP-IND-RCCA secure if for any PPT adversary* $\mathcal{A}$ *the following advantage is negligible in* κ*:*
$\mathsf{Adv}^{\textit{up-ind-rcca}}_{\mathsf{UE},\mathcal{A}}(pp) := \left|\Pr[\mathsf{Exp}^{\textit{up-ind-rcca}}_{\mathsf{UE},\mathcal{A}}(pp,0) = 1] - \Pr[\mathsf{Exp}^{\textit{up-ind-rcca}}_{\mathsf{UE},\mathcal{A}}(pp,1) = 1]\right|$.

```
Experiment $\mathsf{Exp}^{\text{up-ind-rcca}}_{\mathsf{UE},\mathcal{A}}(pp, b)$
    $(sp, k_1, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*) \leftarrow \mathsf{Init}(pp)$
    $(M_0, M_1, state) \leftarrow_{\text{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
    proceed only if $|M_0| = |M_1|$ and $M_0, M_1 \in \mathcal{M}_{sp}$
    $C^* \leftarrow_{\text{R}} \mathsf{UE.Enc}(k_e, M_b)$, $\mathrm{M}^* \leftarrow (M_0, M_1)$, $\mathbf{C}^* \leftarrow \{\mathrm{e}\}$, $e^* \leftarrow e$
    $b' \leftarrow_{\text{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(C^*, state)$
    return $b'$ if $\mathbf{K} \cap \widehat{\mathbf{C}}^* = \emptyset$, i.e. $\mathcal{A}$ did not trivially win. (Else abort.)
```

In the above definition, the global state $(sp, k_e, \Delta_{e-1}, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*)$ is initialized by $\mathsf{Init}(pp)$ as follows:

$\mathsf{Init}(pp)$**:** Returns $(sp, k_1, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}, \mathbf{C}^*)$ where $\mathrm{e} \leftarrow 1$, $sp \leftarrow_{\text{R}} \mathsf{UE.GenSP}(pp)$, $k_1 \leftarrow_{\text{R}} \mathsf{UE.GenKey}(sp)$, $\Delta_0 \leftarrow \perp$, $\mathbf{Q} \leftarrow \emptyset, \mathbf{K} \leftarrow \emptyset$, $\mathbf{T} \leftarrow \emptyset$ and $\mathbf{C}^* \leftarrow \emptyset$.

The list $\mathbf{Q}$ contains "legitimate" ciphertexts that the adversary has obtained through Enc or ReEnc calls. The challenger also keeps track of epochs in which $\mathcal{A}$ corrupted a secret key ($\mathbf{K}$), token ($\mathbf{T}$), or obtained a re-encryption of the challenge-ciphertext ($\mathbf{C}^*$).

Moreover, the oracles given to the adversary are defined as follows:

$\mathsf{Next}()$**:** Runs $k_{\mathrm{e}+1} \leftarrow_{\text{R}} \mathsf{UE.GenKey}(sp)$, $\Delta_{\mathrm{e}} \leftarrow_{\text{R}} \mathsf{UE.GenTok}(k_{\mathrm{e}}, k_{\mathrm{e}+1})$, adds $(k_{\mathrm{e}+1}, \Delta_{\mathrm{e}})$ to the global state and updates the current epoch to $\mathrm{e} \leftarrow \mathrm{e} + 1$.

$\mathsf{Enc}(M)$**:** Returns $C \leftarrow_{\text{R}} \mathsf{UE.Enc}(k_{\mathrm{e}}, M)$ and sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(\mathrm{e}, M, C)\}$.

$\mathsf{Dec}(C)$**:** If $\mathsf{isChallenge}(k_{\mathrm{e}}, C) =$ `false`, it returns $m \leftarrow \mathsf{UE.Dec}(k_{\mathrm{e}}, C)$, else `invalid`.

$\mathsf{ReEnc}(C, i)$**:** Returns C_e iteratively computed as $C_\ell \leftarrow_{\text{R}} \mathsf{UE.ReEnc}(\Delta_{\ell-1}, C_{\ell-1})$ for $\ell = i+1, \ldots, e$ and $C_i \leftarrow C$. It also updates the global state depending on whether the queried ciphertext is the challenge ciphertext or not:

- If $(i, M, C) \in \mathbf{Q}$ (for some m), then set $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(\mathrm{e}, M, C_e)\}$.
- Else, if $\mathsf{isChallenge}(k_i, C) =$ `true`, then set $\mathbf{C}^* \leftarrow \mathbf{C}^* \cup \{\mathrm{e}\}$.

$\mathsf{Corrupt}(\{\mathsf{key}, \mathsf{token}\}, i)$**:** Allows corruption of keys and tokens, respectively:

- Upon input (key, i), the oracle sets $\mathbf{K} \leftarrow \mathbf{K} \cup \{i\}$ and returns k_i.
- Upon input (token, i), the oracle sets $\mathbf{T} \leftarrow \mathbf{T} \cup \{i\}$ and returns Δ_{i-1}.

The isChallenge predicate (used by Dec and ReEnc) is defined as:

$\mathsf{isChallenge}(k_i, C)$: If $\mathsf{UE.Dec}(k_i, C) \in \mathrm{M}^*$, return `true`. Else, return `false`.

To exclude trivial wins, we need to define the set of *challenge-equal epochs* containing all epochs in which the adversary obtains a version of the challenge ciphertext, either through oracle queries or by up/downgrading[11] the challenge ciphertext herself using a corrupted token.

$$\widehat{\mathbf{C}}^* \leftarrow \{e \in \{1,\ldots,e_{\mathsf{end}}\} \mid \mathsf{challenge\text{-}equal}(e) = \mathtt{true}\}$$
and $\mathtt{true} \leftarrow \mathsf{challenge\text{-}equal}(e)$ iff: $(e \in \mathbf{C}^*) \ \vee$
$(\mathsf{challenge\text{-}equal}(e-1) \wedge e \in \mathbf{T}) \ \vee \ (\mathsf{challenge\text{-}equal}(e+1) \wedge e+1 \in \mathbf{T})$

Definition 52 (Perfect Re-encryption [89]). *Let* UE *be an updatable encryption scheme where* UE.ReEnc *is* probabilistic. *We say that re-encryption (of* UE*) is* ***perfect****, if for all $sp \leftarrow_R$* UE.GenSP(pp)*, all keys $k^{\mathrm{old}}, k^{\mathrm{new}} \leftarrow_R$* UE.GenKey$(sp)$*, token $\Delta \leftarrow_R$* UE.GenTok$(k^{\mathrm{old}}, k^{\mathrm{new}})$*, and all ciphertexts C, we have*

$$\mathsf{UE.Enc}(k^{\mathrm{new}}, \mathsf{UE.Dec}(k^{\mathrm{old}}, C)) \overset{\mathrm{dist}}{\equiv} \mathsf{UE.ReEnc}(\Delta, C).$$

In particular, note that $\mathsf{ReEnc}(\Delta, C) = \bot \Leftrightarrow \mathsf{Dec}(k^{\mathrm{old}}, C) = \bot$.

Definition 53 (UP-INT-PTXT [89]). UE *is called UP-INT-PTXT secure if for any PPT adversary $\mathcal{A}$ the following advantage is negligible in κ:*
$\mathsf{Adv}^{\mathit{up\text{-}int\text{-}ptxt}}_{\mathsf{UE},\mathcal{A}}(pp) := \Pr[\mathsf{Exp}^{\mathit{up\text{-}int\text{-}ptxt}}_{\mathsf{UE},\mathcal{A}}(pp) = 1]$.

```
Experiment $\mathsf{Exp}^{\mathsf{up\text{-}int\text{-}ptxt}}_{\mathsf{UE},\mathcal{A}}(pp)$
  $(sp, k_1, \Delta_0, \mathbf{Q}, \mathbf{K}, \mathbf{T}) \leftarrow \mathsf{Init}(pp)$
  $c^* \leftarrow_{\mathrm{R}} \mathcal{A}^{\mathsf{Enc},\mathsf{Dec},\mathsf{Next},\mathsf{ReEnc},\mathsf{Corrupt}}(sp)$
  return 1 if $\mathsf{UE.Dec}(k_{e_{\mathsf{end}}}, c^*) = m^* \neq \bot$ and $(e_{\mathsf{end}}, m^*) \notin \mathbf{Q}^*$,
    and $\nexists e \in \mathbf{K}$ where $i \in \mathbf{T}$ for $i = e$ to $e_{\mathsf{end}}$; i.e. if $\mathcal{A}$ does not trivially win.
```

The oracles provided to the adversary are defined as follows:

Next(), Corrupt($\{\mathsf{key}, \mathsf{token}\}, i$): as in RCCA game

Enc(M): Returns $C \leftarrow_{\mathrm{R}} \mathsf{UE.Enc}(k_{\mathrm{e}}, M)$ and sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(\mathrm{e}, M)\}$.

Dec(C): Returns $m \leftarrow \mathsf{UE.Dec}(k_{\mathrm{e}}, C)$ and sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(\mathrm{e}, M)\}$.

ReEnc(C, i): Returns C_e, the re-encryption of C from epoch i to the current epoch e. It also sets $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(\mathrm{e}, M)\}$ where $M \leftarrow \mathsf{UE.Dec}(k_{\mathrm{e}}, C_{\mathrm{e}})$.

To exclude trivial wins, we define the set $\mathbf{Q}^*$ which contains all plaintexts (and epochs) for which the adversary has received a ciphertext by means of Enc and ReEnc queries or by upgrading a ciphertext herself using a corrupted token.

```
for each $(e, m) \in \mathbf{Q}$:
  set $\mathbf{Q}^* \leftarrow \mathbf{Q}^* \cup (e, m)$, and $i \leftarrow e + 1$
  while $i \in \mathbf{T}$: set $\mathbf{Q}^* \leftarrow \mathbf{Q}^* \cup (i, m)$ and $i \leftarrow i + 1$
```

The adversary trivially wins if her output decrypts to a message m such that (e_{end}, m) is contained in this set or if she has corrupted a secret key and all following tokens, as this allows to create valid ciphertexts for any plaintext.

Moreover, to enable payload integrity along with SNARGs, we will make use of CCA2 secure and re-rerandomizable CPA secure public-key encryption.

Definition 54. *An* asymmetric (or public-key) encryption scheme *consists of three PPT algorithms* (Gen, Enc, Dec) *with the following syntax:*

Key generation. $\mathsf{Gen}(1^\lambda)$ *outputs a public key PK and a secret key SK. We assume that PK defines an efficiently decidable and samplable message space $\mathcal{M} \subseteq \{0,1\}^*$.*

Encryption. $\mathsf{Enc}(PK, M)$ *encrypts a message M under a public key PK to a ciphertext C.*

Decryption. $\mathsf{Dec}(SK, C)$ *decrypts a ciphertext C under a secret key SK to a message $M \in \mathcal{M} \cup \{\mathtt{rej}\}$, where* $\mathtt{rej}$ *is a special symbol that indicates that C was rejected.*

[11]We assume that a token Δ_e also enables *downgrades* of ciphertexts from epoch $e+1$ to epoch e.

We require correctness in the sense that for all λ, all $(PK, SK) \leftarrow \mathsf{Gen}(1^\lambda)$, all $M \in \mathcal{M}$, and all $C \leftarrow \mathsf{Enc}(PK, M)$, we always have $\mathsf{Dec}(SK, C) = M$.

Furthermore, a ciphertext is called valid *(for PK) iff it lies in the range of* $\mathsf{Enc}(PK, \cdot)$*. We say that the scheme has efficiently recognizable ciphertexts iff the set of valid ciphertexts (for a given PK) can be efficiently recognized.*

Finally, we say that the scheme is CPA secure iff no PPT adversary $\mathcal{A}$ can distinguish the following two experiments with non-negligible advantage:

- *$\mathcal{A}$ gets a fresh public key PK, selects two equal-length messages $M_0, M_1 \in \mathcal{M}$, and receives $C^* \leftarrow \mathsf{Enc}(PK, M_0)$.*
- *$\mathcal{A}$ gets a fresh public key PK, selects two equal-length messages $M_0, M_1 \in \mathcal{M}$, and receives $C^* \leftarrow \mathsf{Enc}(PK, M_1)$.*

We say that the scheme is CCA2 secure if the above experiments are indistinguishable even when $\mathcal{A}$ gets access to a decryption oracle $\mathsf{Dec}(SK, \cdot)$ *(with the provision that* $\mathsf{Dec}(SK, \cdot)$ *does not decrypt C^* after C^* is defined).*

Definition 55. *A* rerandomizable asymmetric encryption scheme *is an CPA secure PKE scheme* $(\mathsf{Gen}_{RR}, \mathsf{Enc}_{RR}, \mathsf{Dec}_{RR})$ *with efficiently recognizable ciphertext in the sense of Def. 54 for which a PPT algorithm* Rerand *exists that inputs a public key PK and a ciphertext C, and outputs another ciphertext C'. We require that no PPT adversary $\mathcal{A}$ can distinguish the following two experiments with non-negligible advantage:*

- *$\mathcal{A}$ gets a fresh public key PK, selects C, and receives $C' \leftarrow \mathsf{Rerand}(PK, C)$.*
- *$\mathcal{A}$ gets a fresh public key PK, selects C, and receives $C' \leftarrow \mathsf{Enc}_{RR}(PK, R)$, where R is a fresh random message from the scheme's message space.*

Here, we only quantify adversaries $\mathcal{A}$ that always output valid ciphertexts (in the sense of Def. 54).

We stress that rerandomizability requires that even adversarially generated (but valid) ciphertexts can be rerandomized. An example of a rerandomizable asymmetric encryption scheme is the ElGamal scheme [71]. (The corresponding Rerand homomorphically adds a fresh encryption of the neutral group element to C.)

In the following, we present a variant of the SNARK definition from [77]. We will assume a language $L \subseteq \{0,1\}^*$ (that may depend on the security parameter and additional random choices), and an efficiently computable witness relation R for L. Hence, R takes as input $x \in \{0,1\}^*$ and a potential witness $w \in \{0,1\}^{p(|x|)}$ (for a fixed polynomial p), and gives a binary output. We require that $x \in L \Leftrightarrow (\exists w \in \{0,1\}^{p(|x|)} : R(x, w) = 1)$. We also assume a canonical description of R, e.g., as a Boolean circuit.

Definition 56 (SNARG)**.** *A succinct non-interactive argument (SNARG) for a relation R consists of four PPT algorithms:*

- *Key generation:* $\mathsf{Setup}_{\mathsf{ZK}}(1^\lambda, R)$ *outputs a common reference string CRS and a simulation trapdoor τ.*
- *Proofs:* $\mathsf{Prove}_{\mathsf{ZK}}(R, CRS, x, w)$*, for $R(x, w) = 1$, outputs a proof π.*
- *Verification:* $\mathsf{Vfy}_{\mathsf{ZK}}(R, CRS, x, \pi)$ *outputs a binary verdict.*
- *Simulation:* $\mathsf{Sim}_{\mathsf{ZK}}(R, \tau, x)$ *outputs a simulated proof π.*

We require the following properties:

- *Succinctness: the bitlength $|\pi|$ of π is polynomial in the security parameter λ, and the runtime of* $\mathsf{Vfy}_{\mathsf{ZK}}$ *is polynomial in $\lambda + |x|$.*

- *(Perfect) completeness: for all λ and x, w with $R(x, w) = 1$, it is $\mathsf{Vfy_{ZK}}(R, CRS, x, \mathsf{Prove_{ZK}}(R, CRS, x, w)) = 1$ always.*

- *(Perfect) zero-knowledge: for all λ and x, w with $R(x, w) = 1$, the outputs of $\mathsf{Prove_{ZK}}(R, CRS, x, w)$ and $\mathsf{Sim_{ZK}}(R, \tau, x)$ are identically distributed.*

- *Simulation-soundness: for every PPT $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{c} \mathsf{Vfy_{ZK}}(R, CRS, x, \pi) = 1 \\ \nexists w : R(x, w) = 1 \end{array} \middle| \begin{array}{c} (CRS, \tau) \leftarrow \mathsf{Setup_{ZK}}(1^\lambda, R) \\ (x, \pi) \leftarrow \mathcal{A}^{\mathsf{Sim_{ZK}}(R, \tau, \cdot)}(1^\lambda, R, CRS) \\ \mathsf{Sim_{ZK}} \textit{ never queried with } x \end{array}\right]$$

where the probability is over the random coins of $\mathsf{Setup_{ZK}}$, the random coins of $\mathcal{A}$ and $\mathsf{Vfy_{ZK}}$, and possibly over the choice of the relation R itself.

We note that the simulation-soundness game above is not necessarily efficient (because of the condition $\nexists w : R(x, w) = 1$), but still implied by a property called "simulation-extractability" [90, 78]. Efficient simulation-extractable SNARGs (i.e., SNARKS) can be constructed from knowledge assumptions [78, 107].

C.4.2 Proof Sketches of Further Properties for our UE Scheme

Table C.2: Overview Proof for $LU^{\rightarrow}$, $j < n + 1$

Hybrid	**Description**	**Reduction**
1)	The $LU^{\rightarrow}$ game with challenge bit chosen as 0	
2)	We replace the temporary keys $k_j^{\eta}, k_j^{\gamma}, \Delta_j$ at the honest relay by 0..0 before they are encrypted in E_j (and adapt recognizeOnion to the new header), but still use the real keys for the processing.	CCA2
3)	We let the oracles in Step 7 output a fail, if the challenge E_j is recognized, but other parts of the header differ.	SUF-CMA
4)	We replace the blocks $B_j^1, ..., B_j^{2N-j}$ by $R_1, R_2, \ldots, R_{2N-j}$ with R_i being randomly chosen (and adapt recognizeOnion to the new header), but use the real blocks for the processing.	PRP-CCA
5)	We let the oracles in Step 7 output a fail, if the challenge header η_j is recognized, but the payload does not include the correct plaintext.	UP-INT-PTXT
6)	We let the Proc oracle in Step 7 output the replicated layer $j+1$:$(FormOnion(j+1, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}))$, if the challenge η_j is recognized, the payload matches, and real processing of the given onion would not fail.	Perfect Re-Encryption
7)	We replace the content δ_j by a random string of the same length.	UP-IND-RCCA
8)	We revert the changes made in Game 5).	UP-INT-PTXT
9)	We replace the block B_j^1 by $(\perp, \perp, \perp)$ (and adapt recognizeOnion to the new header).	PRP-CCA
10)	We revert the changes made in Game 3).	SUF-CMA
11)	We revert the changes made in Game 2).	CCA2
12)	We use FormOnion with the parameter of the $b = 1$ case to generate the first challenge onion layer. This is the $LU^{\rightarrow}$ game with challenge bit chosen as 1.	Same behavior except for new draw of randomness

Table C.3: Overview Proof for $LU^{\rightarrow}, j = n+1$

Hybrid	**Description**	**Reduction**
1)	The $LU^{\rightarrow}$ game with challenge bit chosen as 0	
2)	We replace the temporary keys $k_j^{\eta}, k_j^{\gamma}, k_{j=n+1}^{\Delta}$ at the honest relay by 0..0 before they are encrypted in E_j (and adapt recognizeOnion to the new header), but still use the real keys for the processing.	CCA2
3)	We let the oracles in Step 7 output a fail, if the challenge E_j is recognized, but other parts of the header differ.	SUF-CMA
4)	We replace the blocks $B_j^1, ..., B_j^{2N-j}$ by $R_1, R_2, \dots, R_{2N-j}$ with R_i being randomly chosen (and adapt recognizeOnion to the new header), but use the real blocks for the processing.	PRP-CCA
5)	We let oracles in Step 7 output a fail, if the challenge header η_j is recognized, but the payload does not include the correct plaintext.	UP-INT-PTXT
6)	We let the oracles in Step 7 output the replicated layer $j+1$:$(FormOnion(j+1, \mathcal{R}, m^{\leftarrow}, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}))$ for the Reply$(P_H, O, m^{\leftarrow})$ request and we output $(\perp, m)$ for the Proc(P_H, O), if the challenge η_j is recognized, the payload matches, and real processing of the given onion would not fail.	Reply: Same behavior as before, Proc: UE-Correctness
7)	We replace the content δ_j by a random string of the same length.	UP-IND-RCCA
8)	We revert the changes made in Game 5).	UP-INT-PTXT
9)	We replace the block B_j^1 by $\mathsf{PRP.Enc}(\perp, \perp, \perp)$ (and adapt recognizeOnion to the new header).	PRP-CCA
10)	We revert the changes made in Game 6).	SUF-CMA
11)	We revert the changes made in Game 5).	CCA2
12)	We use FormOnion with the parameter of the $b = 1$ case to generate the first challenge onion layer. The $LU^{\rightarrow}$ game with challenge bit chosen as 1	Same behavior except for new draw of randomness

Table C.4: Overview Proof for $LU^{\leftarrow}$, $j^{\leftarrow} = 0$

Hybrid	Description	Reduction
1)	The $LU^{\leftarrow}$ game with challenge bit chosen as 0	
2)	We replace the temporary keys $k^{\eta}_{n+1}, k^{\gamma}_{n+1}, k^{\Delta}_{n+1}$ on the forward path at the honest receiver with $0 \dots 0$ in their encryption for E_{n+1} (and adapt recognizeOnion to the new header), but still use the real keys for the processing.	CCA2
3)	We let the oracles in Step 6 output a fail, if the challenge E_{n+1} is recognized, but other parts of the header differ.	SUF-CMA
4)	We replace the blocks $B^{1}_{n+1}, \dots B^{2N-1}_{n+1}$ by random values when forming the challenge onion (and adapt recognizeOnion to the new header), but use the real block for the processing. (In particular, in this way we get rid of $k^{\Delta}{}^{\leftarrow}_{1}$ and all $k^{\eta}{}^{\leftarrow}_{>j^{\leftarrow}}, k^{\gamma}{}^{\leftarrow}_{>j^{\leftarrow}}, \Delta^{\leftarrow}_{>j^{\leftarrow}}$)	PRP-CCA
5)	We let the keys $k^{\Delta}{}^{\leftarrow}_{1}, k^{\eta}{}^{\leftarrow}_{>j^{\leftarrow}}, k^{\gamma}{}^{\leftarrow}_{>j^{\leftarrow}}, \Delta^{\leftarrow}_{>j^{\leftarrow}}$ and random padding used in Step 6 be freshly chosen and use these for exception 2 of the game.	Games are equivalent
6)	We replace the content $\delta^{\leftarrow}_{1}$ by a random string of the same length during ReplyOnion.	UP-IND-RCCA
7)	We revert the changes made in Game 4).	PRP-CCA
8)	We revert the changes made in Game 3).	SUF-CMA
9)	We revert the changes made in Game 2).	CCA2
10)	The $LU^{\leftarrow}$ game with challenge bit chosen as 1	Same Behavior

Table C.5: Overview Proof for $LU^{\leftarrow}$, $j^{\leftarrow} > 0$

Hybrid	Description	Reduction
1)	The $LU^{\leftarrow}$ game with challenge bit chosen as 0	
2)	We replace the temporary keys $k^{\eta}{}^{\leftarrow}_{j^{\leftarrow}}, k^{\gamma}{}^{\leftarrow}_{j^{\leftarrow}}, \Delta^{\leftarrow}_{j^{\leftarrow}}$ at the honest relay with $0 \dots 0$ in their encryption for $E^{\leftarrow}_{j^{\leftarrow}}$ (and adapt recognizeOnion to the new header) as part of the payload of O_1, but still use the real keys for the processing.	CCA2
3)	We let the oracles in Step 6 output a fail, if the challenge $E^{\leftarrow}_{j^{\leftarrow}}$ is recognized, but other parts of the header differ.	SUF-CMA
4)	We replace all $B^{1}{}^{\leftarrow}_{j^{\leftarrow}}, \dots, B^{2N-1}{}^{\leftarrow}_{j^{\leftarrow}}$ while constructing the header with randomness, but use the real header for $j^{\leftarrow} + 1$ as answer to the corresponding Proc request. Note that replacing all $B^{\leftarrow}_{j^{\leftarrow}}$s results in not including any keys for $> j^{\leftarrow}$ in the earlier header.	PRP-CCA
5)	We let the keys $k^{\Delta}{}^{\leftarrow}_{j^{\leftarrow}+1}, k^{\eta}{}^{\leftarrow}_{>j^{\leftarrow}}, k^{\gamma}{}^{\leftarrow}_{>j^{\leftarrow}}, \Delta^{\leftarrow}_{>j^{\leftarrow}}$, that are used in Step 6 to generate the layer for $j^{\leftarrow} + 1$ when given the challenge header, be freshly chosen and also pick new randomness for the padding of the blocks without path information and use these for exception 2 of the game.	Perfect ReEncryption
6)	We replace the content $\delta^{\leftarrow}_{j}$ with a random string of the same length during ProcOnion at P_H.	UP-IND-RCCA
7)	We revert the changes made in Game 4).	PRP-CCA
8)	We revert the changes made in Game 3).	SUF-CMA
9)	We revert the changes made in Game 2).	CCA2
10)	The $LU^{\leftarrow}$ game with challenge bit chosen as 1	Same Behavior

Table C.6: Overview Proof for TI

Hybrid	Description	Reduction
1)	The $TI^{\leftrightarrow}$ game with challenge bit chosen as 0	
2)	We replace the blocks $B_{j+1}^{2N-(j+1)}, ..., B_{j+1}^{2N-1}$ in Step 5 by random strings $R_{2N-(j+1)}, \dots, R_{2N-1}$ (and adapt recognizeOnion to the new header).	PRP-CCA
3)	We replace the temporary keys $k^{\eta}{}^{\leftarrow}_{j^{\leftarrow}}, k^{\gamma}{}^{\leftarrow}_{j^{\leftarrow}}, \Delta^{\leftarrow}_{j^{\leftarrow}}$ at the honest relay with $0 \dots 0$ in their encryption for $E^{\leftarrow}_{j^{\leftarrow}}$ (and adapt recognizeOnion to the new header) as part of the payload of O_{j+1}, but still use the real keys for the processing.	CCA2
4)	We let the oracles in Step 7 output a fail, if the challenge $E^{\leftarrow}_{j^{\leftarrow}}$ is recognized, but other parts of the header differ.	SUF-CMA
5)	We replace the block $B^{1}{}^{\leftarrow}_{j^{\leftarrow}}$ with a path-end-block and $B^{2}{}^{\leftarrow}_{j^{\leftarrow}}, \dots, B^{(n^{\leftarrow}-j^{\leftarrow}+1)}{}^{\leftarrow}_{j^{\leftarrow}}$ with random blocks in the payload part of the challenge onion representing the backward header.	PRP-CCA
6)	We revert the changes of Game 4).	SUF-CMA
7)	We revert the changes of Game 3).	CCA2
8)	The $TI^{\leftrightarrow}$ game with challenge bit chosen as 1	Same behavior as before

C.4.3 Security of our SNARG-Based Scheme

In this section, we prove that our SNARG-based scheme also realizes the ideal functionality by showing our properties. We start by describing FormOnion for other layers than the first one and continue to show the proofs. As they are similar to the ones in the UE-based solution, we sketch them here.

FormOnion - later layers. FormOnion for $i > 1$ uses the SNARG-trapdoor to create a valid SNARG, encrypts random strings for the ring buffer entries C^j, and creates the other onion parts deterministically as described in the protocol for the current layer. In contrast to the UE-based scheme also the payload is deterministic in the SNARG-based scheme.

Forwards Layer-Unlinkability

Case 1 – Honest Relay ($j < n+1$). We first replace all SNARG-related parts to unlink them from the SNARG information of other layers and also from the secret information included in them. Then we replace the temporary keys of the honest party included in the header, to be able to change the blocks of the header and the payload corresponding to the $b = 1$ case. For the oracles we further need to ensure, that RecognizeOnion does not mistreat any processing of e.g. modified onions. Therefore, we leverage the SNARG properties for the payload protection and the MAC for the header. We provide an overview of the proof in Table C.7.

Case 2 – Honest Receiver ($j = n+1$): We sketch the proof in Table C.8. The steps are the same as for the first case of $LU^{\rightarrow}$, but in Hybrid 10) we need to treat Reply and Proc requests separately. Note that the earlier restrictions on the oracles work both for Reply and Proc requests.

Other properties

We sketch the proofs for the other properties in Table C.9 – C.11.

Backwards Layer-Unlinkability. We distinguish the cases that the honest node is the receiver ($j^{\leftarrow} = 0$) and that it is a backward relay ($j^{\leftarrow} > 0$).

Case 1 – Honest receiver ($j^{\leftarrow} = 0$). The steps are similar to the ones for $LU^{\rightarrow}$ Case 1: We replace the SNARG information and temporary keys of honest routers, before we exclude bad events at the oracle and finally set the header and payload parts to correspond to the $b = 1$ case. Note that for $LU^{\leftarrow}$ we can skip the steps related to the modification of the payload (and SNARG properties). As the forward message is known to the adversary anyways and the backward message (as the final processing) is never given to the adversary, she cannot exploit payload modification at the oracle to break $LU^{\leftarrow}$ in this case.

Case 2 – Honest Relay ($j^{\leftarrow} > 0$). The steps are similar to Case 1 for $LU^{\leftarrow}$.

Repliable Tail-Indistinguishability The steps are similar to Case 2 for $LU^{\leftarrow}$, except that we can skip more steps. For the same reasons as before, we do not need the payload protection in $TI^{\leftrightarrow}$. Further, due to the use of FormOnion (for layers > 1) the ring buffer entries $C_{>j+1}$ do not encrypt any sensitive information, but only random bits and thus do not need to be replaced in the beginning. Finally, the adversary does not obtain any leakage related to k_j^{η} and thus the blocks in the forward header (Step 3)) can be replaced right away.

Table C.7: Overview Proof for $LU^{\rightarrow}$, $j < n+1$

Hybrid	Description	Reduction
1)	The $LU^{\rightarrow}$ game with challenge bit chosen as 0	
2)	We simulate the SNARGs for the challenge onion.	SNARG simulatability
3)	We replace the first ring buffer entry C_1^1 with a fresh encryption of `sim` for the special bitstring `sim`.	CPA/rerand. of PKM
4)	We replace the ring buffer elements C_{j+1}^i for all i with fresh encryptions of random strings (not `sim`) [as FormOnion does for layers $i > 1$].	CPA/rerand. of PKM
5)	We replace the temporary keys $k_j^\eta, k_j^\gamma, k_j^\delta$ at the honest relay by 0..0 before they are encrypted in E_j (and adapt recognizeOnion to the new header), but still use the real keys for the processing.	CCA2
6)	We let the oracles in step 7 output a fail, if the challenge E_j is recognized, but other parts of the header differ.	SUF-CMA
7) & 8)	We replace the blocks $B_j^1, ..., B_j^{2N-j}$ by $(\mathtt{sim}, \mathtt{sim})$, $R_2, \ldots, R_{N-j}$ with R_i being randomly chosen (and adapt recognizeOnion to the new header), but use the real blocks for the processing.	PRP-CCA
9)	We let oracles in step 7 output a fail, if the challenge header η_j is recognized, but the payload differs.	SNARG simulation soundness
10)	We let the oracles in step 7 output the replicated layer $j+1$:(FormOnion($j+1, \mathcal{R}, m, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}$)), if the challenge η_j is recognized, the payload matches, and real processing of the given onion would not fail.	Same behavior due to definition of recognition and forming of later layers
11)	We replace the content δ_j by a random string of the same length.	PRP-CCA
12)	We revert the changes made in Game 9).	SNARG simulation soundness
13) & 14)	We replace the blocks $B_j^1, ..., B_j^{2N-j}$ by $(\perp, \perp, \perp)$, $R_2, \ldots, R_{2N-j}$ with R_i being randomly chosen (and adapt recognizeOnion to the new header).	PRP-CCA
15)	We revert the changes made in Game 6).	SUF-CMA
16)	We revert the changes made in Game 5).	CCA2
17)	We revert the changes made in Game 3): The ring buffer entry C_1^1 now includes the sender info as in the $b = 1$ case.	CPA/rerand. of PKM
18)	We use FormOnion with the parameter of the $b = 1$ case to generate the first challenge onion layer.	Same behavior except for new draw of randomness
19)	The $LU^{\rightarrow}$ game with challenge bit chosen as 1	SNARG simulatability

Table C.8: Overview Proof for $LU^{\rightarrow}$, j=n+1

Hybrid	Description	Reduction
1)	The $LU^{\rightarrow}$ game with challenge bit chosen as 0	
2)	We simulate the SNARGs for the challenge onion.	SNARG simulatability
3)	We replace the first ring buffer entry C_1^1 with a fresh encryption of `sim` for the special bitstring `sim`.	CPA/rerand. of PKM
4)	We replace the ring buffer elements C_{j+1}^i for all i with fresh encryptions of random strings (not `sim`) [as FormOnion does for layers $i > 1$].	CPA/rerand. of PKM
5)	We replace the temporary keys $k_j^{\eta}, k_j^{\gamma}, k_j^{\delta}$ at the honest relay by 0..0 before they are encrypted in E_j (and adapt recognizeOnion to the new header), but still use the real keys for the processing.	CCA2
6)	We let the oracle in step 7 output a fail, if the challenge E_j is recognized, but other parts of the header differ.	SUF-CMA
7) & 8)	We replace the blocks $B_j^1, ..., B_j^{2N-j}$ by $(\texttt{sim}, \texttt{sim})$, $R_2, \ldots, R_{2N-j}$ with R_i being randomly chosen (and adapt recognizeOnion to the new header), but use the real blocks for the processing.	PRP-CCA
9)	We let oracle in step 7 output a fail, if the challenge header η_j is recognized, but the payload differs.	SNARG simulation soundness
10)	We let the oracle in step 7 output the replicated layer $j+1$:$(FormOnion(j+1, \mathcal{R}, m^{\leftarrow}, \mathcal{P}^{\rightarrow}, \mathcal{P}^{\leftarrow}, (PK)_{\mathcal{P}^{\rightarrow}}, (PK)_{\mathcal{P}^{\leftarrow}}))$ for $\mathsf{Reply}(P_H, O, m^{\leftarrow})$ and we output $(\perp, m)$ for $\mathsf{Proc}(P_H, O)$, if the challenge η_j is recognized, the payload matches, and real processing of the given onion would not fail.	Same behavior due to definition of recognition and forming of later layers
11)	We replace the content δ_j by a random string of the same length.	PRP-CCA
12)	We revert the changes made in Game 10).	SNARG simulation soundness
13) & 14)	We replace the blocks $B_j^1, ..., B_j^{2N-j}$ by $(\perp, \perp, \perp)$, $R_2, \ldots, R_{2N-j}$ with R_i being randomly chosen (and adapt recognizeOnion to the new header).	PRP-CCA
15)	We revert the changes made in Game 6).	SUF-CMA
16)	We revert the changes made in Game 5).	CCA2
17)	We revert the changes made in Game 3): The ring buffer entry C_1^1 now includes the sender info as in the $b = 1$ case.	CPA/rerand. of PKM
18)	We use FormOnion with the parameter of the $b = 1$ case to generate the first challenge onion layer.	Same behavior except for new draw of randomness
19)	The $LU^{\rightarrow}$ game with challenge bit chosen as 1	SNARG simulatability

Table C.9: Overview Proof for $LU^{\leftarrow}$, $j^{\leftarrow} = 0$

Hybrid	**Description**	**Reduction**
1)	The $LU^{\leftarrow}$ game with challenge bit chosen as 0	
2)	We simulate the SNARGs for the challenge onion.	SNARG simulatability
3)	We replace the first ring buffer entry C_1^1 with a fresh encryption of `sim` for the special bitstring `sim`.	CPA/rerand. of PKM
4)	We replace the first ring buffer entry $C^{1}{}^{\leftarrow}_{j^{\leftarrow}+1}$ with a fresh encryption of `sim` for the special bitstring `sim`.	CPA/rerand. of PKM
5)	We replace the temporary keys $k^{\eta}_{n+1}, k^{\gamma}_{n+1}, k^{\delta}_{n+1}$ on the forward path at the honest receiver with $0 \dots 0$ in their encryption for E_{n+1} (and adapt recognizeOnion to the new header), but still use the real keys for the processing.	CCA2
6)	We let the oracle in step 6 output a fail, if the challenge E_{n+1} is recognized, but other parts of the header differ.	SUF-CMA
7)	We replace the block $B^1_{n+1}, \dots, B^{2N-1}_{n+1}$ by a random blocks when forming the challenge onion (and adapt recognizeOnion to the new header), but use the real block for the processing. (In particular, in this way we get rid of all $k^{\eta}{}^{\leftarrow}_{>j^{\leftarrow}}, k^{\gamma}{}^{\leftarrow}_{>j^{\leftarrow}}, k^{\delta}{}^{\leftarrow}_{>j^{\leftarrow}}$)	PRP-CCA
8)	We let the keys $k^{\eta}{}^{\leftarrow}_{>j^{\leftarrow}}, k^{\gamma}{}^{\leftarrow}_{>j^{\leftarrow}}, k^{\delta}{}^{\leftarrow}_{>j^{\leftarrow}}$ used in step 6 be freshly chosen by $P^{\leftarrow}_{j^{\leftarrow}}$.	Games are equivalent
9)	We replace the content $\delta^{\leftarrow}_1$ by a random string of the same length during ReplyOnion.	PRP-CCA
10)	We revert the changes made in Game 7).	PRP-CCA
11)	We revert the changes made in Game 6).	SUF-CMA
12)	We revert the changes made in Game 5).	CCA2
13)	We revert the changes made in Game 4): $C^{1}{}^{\leftarrow}_{j^{\leftarrow}+1}$ contains now the information of $P^{\leftarrow}_{j^{\leftarrow}}$ as sender.	CPA/rerand. of PKM
14)	We revert the changes made in Game 3).	CPA/rerand. of PKM
15)	The $LU^{\leftarrow}$ game with challenge bit chosen as 1	SNARG simulatability

Table C.10: Overview Proof for $LU^{\leftarrow}$, $j^{\leftarrow} > 0$

Hybrid	Description	Reduction
1)	The $LU^{\leftarrow}$ game with challenge bit chosen as 0	
2)	We simulate the SNARGs for the challenge onion.	SNARG simulatability
3)	We replace the first ring buffer entry C_1^1 with a fresh encryption of `sim` for the special bitstring `sim`.	CPA/rerand. of PKM
4)	We replace the first ring buffer entry ${C^1}^{\leftarrow}_{j^{\leftarrow}+1}$ with a fresh encryption of `sim` for the special bitstring `sim`.	CPA/rerand. of PKM
5)	We replace the temporary keys ${k^\eta}^{\leftarrow}_{j^{\leftarrow}}, {k^\gamma}^{\leftarrow}_{j^{\leftarrow}}, {k^\delta}^{\leftarrow}_{j^{\leftarrow}}$ at the honest relay with $0\dots0$ in their encryption for $E^{\leftarrow}_{j^{\leftarrow}}$ (and adapt recognizeOnion to the new header) as part of the payload of O_1, but still use the real keys for the processing.	CCA2
6)	We let the oracle in step 6 output a fail, if the challenge $E^{\leftarrow}_{j^{\leftarrow}}$ is recognized, but other parts of the header differ.	SUF-CMA
7)	We replace all ${B^1}^{\leftarrow}_{j^{\leftarrow}}, ..., {B^{2N-1}}^{\leftarrow}_{j^{\leftarrow}}$ while constructing the header with randomness, but use the real header for $j^{\leftarrow}+1$ as answer to the corresponding Proc oracle request. Note that replacing all $B^{\leftarrow}_{j^{\leftarrow}}$s results in not including any keys for $> j^{\leftarrow}$ in the earlier header.	PRP-CCA
8)	We let the keys ${k^\eta}^{\leftarrow}_{>j^{\leftarrow}}, {k^\gamma}^{\leftarrow}_{>j^{\leftarrow}}, {k^\delta}^{\leftarrow}_{>j^{\leftarrow}}$ used in step 6 be freshly chosen by $P^{\leftarrow}_{j^{\leftarrow}}$.	Games are equivalent
9)	We replace the content $\delta^{\leftarrow}_{j^{\leftarrow}}$ with a random string of the same length during FormOnion.	PRP-CCA
10)	We revert the changes made in Game 7).	PRP-CCA
11)	We revert the changes made in Game 6).	SUF-CMA
12)	We revert the changes made in Game 5).	CCA2
13)	We revert the changes made in Game 4): ${C^1}^{\leftarrow}_{j^{\leftarrow}+1}$ contains now the information of $P^{\leftarrow}_{j^{\leftarrow}}$ as sender.	CPA/rerand. of PKM
14)	We revert the changes made in Game 3).	CPA/rerand. of PKM
15)	The $LU^{\leftarrow}$ game with challenge bit chosen as 1	SNARG simulatability

Table C.11: Overview Proof for TI

Hybrid	Description	Reduction
1)	The $TI^{\leftrightarrow}$ game with challenge bit chosen as 0	
2)	We simulate the SNARGs for the challenge onion.	SNARG simulatability
3)	We replace the blocks $B_{j+1}^{2N-(j+1)}, ..., B_{j+1}^{2N-1}$ in step 5 by random strings $R_{n-j+2}, \dots, R_{N-1}$ (and adapt recognizeOnion to the new header).	PRP-CCA
4)	We replace the temporary keys ${k^\eta}^{\leftarrow}_{j^{\leftarrow}}, {k^\gamma}^{\leftarrow}_{j^{\leftarrow}}, {k^\delta}^{\leftarrow}_{j^{\leftarrow}}$ at the honest relay with $0\dots0$ in their encryption for $E^{\leftarrow}_{j^{\leftarrow}}$ (and adapt recognizeOnion to the new header) as part of the payload of O_{j+1}, but still use the real keys for the processing.	CCA2
5)	We let the oracle in step 7 output a fail, if the challenge $E^{\leftarrow}_{j^{\leftarrow}}$ is recognized, but other parts of the header differ.	SUF-CMA
6)	We replace the block ${B^1}^{\leftarrow}_{j^{\leftarrow}}$ with a path-end-block and ${B^2}^{\leftarrow}_{j^{\leftarrow}}, \dots, {B^{(n^{\leftarrow}+1-j^{\leftarrow})}}^{\leftarrow}_{j^{\leftarrow}}$ with random blocks in the payload part of the challenge onion representing the backward header.	PRP-CCA
7)	We replace the first ring buffer entry C_{j+1}^1 with the information of P_j as sender.	CPA/rerand. of PKM
8)	We revert the changes of Game 5).	SUF-CMA
9)	We revert the changes of Game 4).	CCA2
10)	The $TI^{\leftrightarrow}$ game with challenge bit chosen as 1	SNARG simulatability

C.4.4 Performance

UE-Based Scheme - Performance

Considering the requirements of our UE-based scheme, we are currently only aware of a single suitable UE scheme which is a construction by Klooss, Lehmann, and Rupp [89] based on (the malleability of) Groth-Sahai proofs. Unfortunately, instantiating our protocol with their UE scheme leads to payload parts of the onion which are comparatively large: Their underlying algebraic structure is a pairing-based group setting $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. To encrypt a single $\mathbb{G}_1$-element, the payload part contains 58 $\mathbb{G}_1$-and 44 $\mathbb{G}_2$-elements. For realistic group (bit)sizes of, say, $|\mathbb{G}_1| = 256$ and $|\mathbb{G}_2| = 512$, we obtain a payload size of about 4.5 kilobytes for 256 bits of communicated message. The header part of the onion is about half as large for small pathlengths, and using conventional state-of-the-art building blocks, a full onion (including header and payload) comes out at about $4.5 + N$ kilobytes, where N is the maximal length of a path, i.e., the number of hops between sender and receiver (see below for details). Processing an onion at a relay is dominated by the cost to perform the re-encryption of the payload which requires about 110 $\mathbb{G}_1$- and 90 $\mathbb{G}_2$-exponentiations [89].

Details: All sizes in the following calculation are in bits.

- Kurosawa-Desmedt [102] as the CCA2-secure PKE: $|PK| = 512$, $|C| = |M| + 640$.
  - Remark: We count only user-specific parts in *pk*, the rest can be pushed into global public parameters. *pk* contains 2 group elements, and C contains 2 group elements and an authenticated encryption of M.
- SHA-3 [141] as hash: $|P| = |\gamma| = 256$ (for HMAC-based MACs with $|k_i^\gamma| = 128$)
  - Remark: We count an identity as the size of a hash value (like previous approaches).
- AES-128 [1] as symmetric encryption scheme: $|k_i^\eta| = 128$
- The NYUAE scheme from [89] for the payload: $|k_i^\Delta| = 2560$, $|\Delta_i| = 1536$, $|\delta_i = 37376|$
  - Remark: We consider a pairing-based group setting $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with $p = |\mathbb{G}_1| = 256$ and $|\mathbb{G}_2| = 512$. A key consists of 4 $\mathbb{F}_p$, 2 $\mathbb{G}_1$, and 2 $\mathbb{G}_2$ elements, whereas a token consists of 4 $\mathbb{F}_p$, and 2 $\mathbb{G}_1$ elements. As header ciphertexts should not leak whether they contain a key or token, we need to pad to the maximum of both. The payload consists of about 58 $\mathbb{G}_1$, and 44 $\mathbb{G}_2$ elements.

Hence:

- $|E_i| = 2 \cdot 128 + 20 \cdot 128 + 640 = 3456$,
- $|\gamma_i| = 256$,
- $|B_i^j| = 256 + 3456 + 256 = 3968$:
- $|\eta_i| = (2N - 1) \cdot 3968 + 3456 + 256 = (2N - 1) \cdot 3968 + 3712$.
- In total: $|O_i| = |\eta_i| + |\delta_i| = (2N - 1) \cdot 3968 + 3712 + 37376$ bits.

A realistic value for N (maximal path length) can be $N = 3$ or $N = 4$, which leads to an onion size overhead (over $|m|$) of about 7.5 kbytes, resp. 8.5 kbytes.

SNARG-Based Scheme - Performance

Our SNARG-based onions are in fact smaller (for small pathlengths N) than the ones from our UE-based protocol. Using the SNARKs of Groth and Maller [78] (and state-of-the-art conventional building blocks), we obtain onions with an additive overhead (over the message size) of $128N^2 + 448N + 192(2N-1) + 160$ bytes (see below for details). The perhaps surprising quadratic term in the maximal pathlength N stems from the fact that we require additional encryptions of *all* previous onion headers to enable a recursive extraction of previous onion states.

However, due to our somewhat complex SNARG language, we expect that the actual processing time of our SNARG-based approach (which involves constructing SNARG proofs at each processing step) will be considerably higher than the one from our UE-based protocol.

Details: All sizes in the following calculation are in bits.

- As in the UE-based protocol: Kurosawa-Desmedt [102] as the CCA2-secure PKE, SHA-3 [141] as hash and AES-128 [1] as symmetric encryption scheme.
- SNARKs of Groth and Maller [78]: $|\pi| = 1024$
  - Remark: This building block operates in a pairing setting with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where we can assume. In this setting, $\mathbb{G}_1$-, resp. $\mathbb{G}_2$-elements can be set to have 256, resp. 512 bits.
- (Multi-generator-)ElGamal as the rerandomizable PKE: $|C| = |M| + 256$
  - Remark: We represent a plaintext string of $\ell \cdot 256$ bits as a vector of ℓ group elements $m_1, \dots, m_\ell$ (of a suitably-sized group $\mathbb{G}$) and can then set $C = (g^r, h_1^r m_1, \dots, h_k^r m_k)$ for random r and public key elements $h_1, \dots, h_k$. Hence $|pk| = \ell \cdot 256$. In our setting, $\ell = 4N + 9$ since we encrypt $(N+2) \cdot 1024 + 256$ bits (see below). Rerandomization adds (componentwise) an encryption of $(1_{\mathbb{G}})^\ell$.

Hence:

- $|E_i| = 3 \cdot 128 + 640 = 1024$,
- $|\gamma_i| = 256$,
- $|B_i^j| = 256 + 1024 + 256 = 1536$:
- $|\eta_i| = (2N-1) \cdot 1536 + 1024 + 256 = (2N-1) \cdot 1536 + 1280$.
- C_i^j: These are ElGamal ciphertexts for messages of size $1024 + N \cdot 1024 + 1024 + 256 = (N+2) \cdot 1024 + 256$ each. This size calculation uses that
  - all C_i^j have the same size, and that
  - C_i^1 has a shorter actual (i.e., in its unpadded form) plaintext than C_i^j $(j > 1)$, since the m encrypted in C_i^1 can be made implicit and reconstructed from R and the encrypted payload in O_1.
  - We do not count the "proc" string at the beginning of each C_i^j $(j > 1)$, since this string can be encoded as a single bit.
- σ_i contains N C_i^j's and N SNARGs: $|\sigma_i| = N \cdot (1024 + N + 2) \cdot 1024 + 256 + 256) = N^2 \cdot 1024 + N \cdot 3584$.
- $|\delta_i| = |m|$ (encrypted in-place)
- In total: $|O_i| = |\eta_i| + |\sigma_i| + |\delta_i| = |m| + N^2 \cdot 1024 + N \cdot 3584 + (2N-1) \cdot 1536 + 1280$ bits.

A realistic value for N (maximal path length) can be $N = 3$ or $N = 4$, which leads to an onion size overhead (over $|m|$) of about 3kbytes, resp. 5kbytes.+

Performance in comparison to Ando and Lysyanskaya.

While Ando and Lysyanskaya do not provide concrete efficiency calculations, conceptually their and our (time and space) overhead are similar *except* for the parts related to updatable encryption, resp. SNARGs. For realistic security parameters, these parts dominate the header overhead. This is the price one has to pay for preventing the malleability attack while making reply onions indistinguishable from request onions as desired by practical protocols like HORNET [43].

After all, this is the first approach providing immunity in this strong sense, and while we do not claim optimality of our constructions, we are convinced they are the basis for a real-world improvement in communication privacy.

D. Details on Related Areas

D.1. Proximity Tracing Applications

D.1.1 Examples

For our examples horizontal lines symbolize the end of a batch, $t > 2$ and user A is corrupt.

Proximity Tracing Indistinguishability ($P{-}IND$)

Scenario 0	Scenario 1
m(A,B)	m(A,C)
m(C,E)	⋄
m(A,C)	m(A,D)
m(A,C)	m(A,E)
i(B)	i (E)
i(D)	⋄

Infection Indistinguishability ($I{-}IND$)

Scenario 0	Scenario 1
m(A,B)	m(A,B)
m(A,B)	m(A,B)
m(A,C)	m(A,C)
i(B)	i (C)
i(C)	⋄

Infected Indistinguishability ($I \setminus |I|{-}IND$)

Scenario 0	Scenario 1
m(A,B)	m(A,B)
m(A,B)	m(A,B)
m(A,C)	m(A,C)
i(B)	i (C)

Meeting Indistinguishability ($M{-}IND$)

Scenario 0	Scenario 1
m(A,B)	m(A,C)
m(A,D)	m(A,D)
m(C,D)	$\diamond$
i(B)	i (B)
i(C)	i (C)

Healthy meeting Indistinguishability ($M\backslash I{-}IND$)

Scenario 0	Scenario 1
m(A,B)	m(A,B)
m(A,C)	m(A,D)
m(C,D)	$\diamond$
i(B)	i (B)

Remote Colocation Indistinguishability ($R{-}IND$)

Scenario 0	Scenario 1
m(A,C)	m(A,C)
m(B,C)	m(C,D)
i(B)	i (B)

Remote Healthy Colocation Indistinguishability ($R\backslash I{-}IND$)

Scenario 0	Scenario 1
m(A,C)	m(A,C)
m(B,C)	m(B,C)
m(E,C)	m(C,D)
i(B)	i (B)

D.1.2 Possible Extensions

Investigating Unavoidable Leakage. A relaxation of the functionality or assumptions of the used techniques would lead to other unavoidable leakage. Especially that the frequency of meetings for corrupted users and the meetings with corrupted are equal is inherent in the underlying assumption that broadcasting techniques for some temporary identifiers are used, which most important proposals do, but it is not inherent in the desired functionality. Further, the risks embedded in the unavoidable leakage can be investigated by removing it from the definition and analyzing the resulting adversarial advantage in any assumed setting. The advantage thus allows to conclude the probability with which a case breaking the goal occurs in the analyzed setting.

Different Levels of Meeting Intensity and Notifications. Our meeting event can be extended by a parameter that specifies the level of intensity, like e.g. the duration of a meeting or the physical distance between the two people. For the notions one would then again need to specify which meeting intensity each of the properties considers.

More Timing Information for the Infection. If health workers define the point of being contagious differently than t time units before the detection, this information can be modeled as part of the infection event and the notions that consider the calculation of warnings can be adapted to make statements about the corresponding times.

Adding Locations. Challenging the adversary to distinguish scenarios based on the location at which users are is certainly interesting to analyze for approaches that collect some form of location data. For the approaches that do not collect such location data to start with, such notions are trivially achieved. Modeling locations such that only reasonable inputs can be given by the adversary makes this a challenging task for future work, e.g. the locations of the same honest user in consecutive batches should be close, people that are sufficiently close should have a meeting event and different dimensions of location (GPS coordinate and height) possibly have to be considered.

Epochs of Reused Pseudonyms. Many protocols are based on epochs, in which one pseudonym is used. The pseudonym is only changed for the next epoch. If all users synchronize their epoch changes (epochs are based on a global timer), this is easy to model. We can simply understand an epoch as the encapsulation of the batches that correspond to its time and extend properties as needed to consider events of this larger time span. If however all users change epochs based on their local time, the adaptation of the properties is no longer straightforward and should be subject of future work.

Friends, Family and People with same Routine. If one wants to model that no protection from friends and family is needed (as one might assume they share the information of being infected anyways), one can restrict the scenarios to only differ in infected users that do not meet more often than one expects people with the same routine to meet. Similarly the additional requirement that only people with the same routine differ in their infected status in the two scenarios, can be added as restriction (if one assumes that only regular met people are at a privacy risk as completely random encounters can most likely not be identified anyways [152]).

Assumed Behavior after Detection of Infection. We assumed that users with a detected infection strictly self-isolate and that a positive test is not projected forward throughout quarantine and thus is not triggering further alerts. Without this assumptions further attacks are imaginable, like testing the infection status by provoking meetings with the victim, and changes on the properties would be necessary, e.g. M_i would also need to ensure that the meeting events of an infected user *after* the infect was detected are equal in both scenarios for the corresponding time span.

Sybil-Attacks. An easy attack is that an adversarial user ensures to meet only one user (by blocking later exchange of colocation data, i.e. temporal pseudonyms). If this adversarial user receives a warning, the adversary knows that the one met user is infected. If the adversary can easily create many identities, she can do this attack for many victims. As a countermeasure against this attack, sybil-protection measures might be introduced into a protocol. The effect of such measures can be translated into our model by limiting the number of clients that the adversary is allowed to corrupt. However, to also observe an effect on the notions, they have to be adapted such that an adversary cannot pick arbitrarily many differences between the scenarios. Such a quantitative restriction of the notions can be done similar to the number of challenges in Appendix A.1.3.

Relaxation of Advantage Definition. As done before for anonymous communication networks, in some cases the strict advantage definition is too strong and allowing to distinguish with a non-negligible, but still very small additive probability is acceptable. Therefore, the advantage definition can be adapted similar to the (ϵ, δ)-differential privacy definition. However, the properties can be kept the same and the same hierarchy follows.

Tracing of Users. A typical goal is that the colocation events of a user at different times cannot be linked to each other, as otherwise the complete movement of the user could be traced by an adversary. To capture this protection in a notion, we need a property that ensures that in the first scenario the same user was met twice and in the second two different users were met. Example (A corrupted):

Scenario 0	Scenario 1
m(A,B)	m(A,B)
m(A,B)	m(A,C)

However as we do not want the adversary to win the scenarios by identifying the user *only* in the first meeting, we need to ensure that this alternative is used in 50% of the games:

Scenario 0	Scenario 1
m(A,C)	m(A,C)
m(A,C)	m(A,B)

Formally, such a property requires small extensions to the game, but can be constructed similar to T_S in Section 3.3. Note that depending on how close these two batches are, this idea is able to capture the leakage due to reused pseudonyms (if the batches are so close that still the same pseudonym is used, e.g. directly after each other as in the example above) or to cover the leakage due to linkable pseudonyms (if we force the adversary to compare batches that are in different pseudonym epochs).

More Notions and Properties. We limited this consideration to a first small set of useful notions. However, we expect further interesting properties to be revealed during future work on protocol analysis. One such example might be the notion that permits to learn the number of infected users that one has encountered, which likely expresses a difference between centralized and decentralized approaches. Some additional properties are already listed informally in the following paragraphs:

Infections. One protectable property only concerns infection events:

Set of infected users U_i: Who is infected (all or a subset of the infected users) is known or not known to the adversary.

Meetings. Properties that only concern user meetings and no infections: With regard to single users:

Complete meeting info per user CI_u: This information contains a list of all meeting events this user was in. This includes who a user met, how often and in which order.

Meeting sets M: This only includes which other user each user met, but not how often or in which order.

Frequency in users Q_u: This specifies for any user, how many users she met.

Frequency per user Q: This specifies for any user, how often she met any other user.

Frequency in meetings Q_m: This contains for any user, the number of meeting events she had.

With regard to multiple users:

Complete meeting information CI: This lists all meeting events that happened. Compared to CI_u this also allows to infer the order of meetings from different users.

Partitioning P: This specifies the subset of users, in which each user met at least one other user of the set (connected components in the meeting graphs) in a given time span.

Histogram H: This specifies how many users met how many users (e.g. 10 users met 1 user each, 5 users 2 etc.).

Combined. Properties that arise from meeting infected users: With regard to single users:

Complete meeting info restricted to infected $CI_u^{\to I}$: This lists all meetings with infected users the considered user had. It includes which infected user she met, how often and in which order.

Meeting set restricted to infected users $M^{\to I}$: This specifies which infected users a user met.

Frequency in infected users $Q_u^{\to I}$: This specifies how many infected users a user met.

Frequency per infected user $Q^{\to I}$: This specifies how often a user met a certain infected user, for each infected user.

Frequency in meetings with infected users $Q_m^{\to I}$: This specifies how many meeting events with infected users each user had.

With regard to multiple users:

Complete information restricted to infectes users $CI^{\to I}$: This lists all meeting events with infected users involved.

Histogram restricted to infected users. $H_m^{\to I}$: This specifies how many users met how many infected users.

In short, we restrict the properties above to only take the infected users into account $X^{\to I}$ in the combined properties.

We can also turn this around to describe which infected user met which other users and restrict the statistics on the infected users $X^{I\to}$. With regard to single users:

Complete meeting info for infected $CI_u^{I\to}$ lists all events in which an infected user is involved (as $CI_u^{\to I}$).

Meeting sets for infected $M^{I\to}$ specifies which users any infected user met (not the same as $M^{\to I}$, but includes same information).

Frequency in user per infected $Q_u^{I\to}$ specifies how many users an infected user met (different information than $Q_u^{\to I}$).

Frequency per user for infected $Q^{I\to}$ specifies how often an infected user met another user (different information than $Q^{\to I}$).

Frequency in meetings for infected $Q_m^{I\to}$ specifies how many meeting events with users each infected user had (different information than $Q_m^{\to I}$).

Similarly, we can restrict this for contact of infected users under each other: $X^{I\to I}$.

Note on other user sets. Similarly, we can restrict these properties on any other subset of users. Especially, the corrupted users are useful in the unavoidable leakage: $X^{\to C}$, $X^{C\to}$ and $X^{C\to C}$.

Table D.1: An initial approximation for proposed protocols (formal proofs, and in many cases even exact protocol specifications do not exist at the time of writing)

Proposal	**Adversary**	$P{-}IND$	$I{-}IND$	$I \setminus \|I\|{-}IND$	$M{-}IND$	$M \setminus I{-}IND$	$R{-}IND$	$R \setminus I{-}IND$
central tracing server	client(s)	✓	✓	✓	✓	✓	✓	✓
	server	X	X	X	X	X	X	X
NTK/ROBERT	client(s)	✓	✓	✓	✓	✓	✓	✓
	server	X	X	X	X	✓	X	✓
DESIRE	client(s)	X	X	X	X	✓	✓	✓
	server	X	X	X	X	✓	X	✓
ConTraCorona	client(s)	X	X	X	X	✓	✓	✓
	server	X	X	X	X	✓	X	✓
Canetti	client(s)	X	X	X	X	✓	✓	✓
	server	X	X	✓	✓	✓	✓	✓
MIT (PSI)	client(s)	X	X	X	X	✓	✓	✓
	server	X	X	X	X	✓	X	✓
DP3T	client(s)	X	X	X	X	✓	✓	✓
	server	X	X	✓	✓	✓	✓	✓

Table D.2: Information about location and pseudonyms in the protocols

Proposal	**Adversary**	**Location?**	**Pseudonyms trivially linkable? (for infected)**
central tracing server	client(s)	no	never
	server	exact	forever
NTK/ROBERT	client(s)	no	n^{th} part of day (15min)
	server	no	from contagiousness until quarantine
DESIRE	client(s)	no	never
	server	no	never
ConTraCorona	client(s)	no	never
	server	no	never
Canetti	client(s)	no	never
	server	no	from contagiousness until quarantine
MIT (PSI)	client(s)	region	no pseudonym
	server	region	no pseudonym
DP3T (Low-cost)	client(s)	no	from contagiousness until quarantine, n^{th} part of day (no infection)
	server	no	from contagiousness until quarantine
DP3T (Unlinkable)	client(s)	no	n^{th} part of day
	server	no	from contagiousness until quarantine
DP3T (Hybrid)	client(s)	no	m^{th} part of day ($m < n$), 2 or 4 hours recommended
	server	no	from contagiousness until quarantine

D.1.3 Notes on existing approaches

Investigating a protocol regarding a notion is only meaningful under the assumption of a certain adversary and regarding different adversaries the results for the same notion and protocol can differ. For this first short review we are concerned with the adversary model that corrupts only clients and the one that corrupts only the server.

For our educated guess, we utilize existing implications between the notions, i.e. if we expect that a strong notion is achieved this includes that all weaker, implied notions are achieved as well. Further, we use a similar logic for adversaries; if a weak adversary (e.g. one corrupting only one client) is expected to break the notion, a strictly stronger adversary (e.g. one corrupting multiple clients) is, too.

We are limiting our short review of the state of the art at the time of writing (April 2020) in three ways:

Reusing of Pseudonyms ignored. As long as the broadcast pseudonym is not changed, multiple meetings can trivially be linked (see Appendix D.1.2 for a corresponding notion). This means that for this time span the user's meetings are traceable, if the adversary is able to observe the area, or at least manages to incur several encounters. As the information how long a pseudonym is reused is important, we include this parameter in Table D.2. Other than that, we ignore this attack vector for the moment, i.e. we assumed that any batch uses new unlinkable pseudonyms, to get a feeling for the protection introduced by the other design decisions[1].

Location Information ignored. Early protocols considered leveraging location information to calculate the at-risk users. As this of course is an important private information, we include it in our table as well. Other than that, we however ignore this information at the moment. In terms of our security games we thus assume that the locations do not differ between the scenarios, i.e. the same location is being used for the corresponding meeting of the second scenario as the one used in the first scenario. Also the location of one user in one batch does not change and locations of the same user in two consecutive batches are assumed to be reasonably close.

Educated Guess. We limit our investigation to provide an educated guess of reasons, why some approaches systematically may, or may not achieve specific notions. A summary of this guess can be found in Table D.1 (the results for one and multiple clients are expected to be identical). As we believe that providing the formalization of privacy goals as notions is more pressing at this point in time, to help the various developers understand and express their goals and approaches better, we leave an in-depth analysis of the protocols and the corresponding proofs for future work.

Centralized location tracing server[2]

Clients. This approach may protect perfectly from corrupted clients as they do not have to share information with other clients, but only with the server.

Server. The server learns the locations of all users at all times. Thus, it learns even more information than the colocation events that we modeled here.

Notions. $P-IND$ should hold for multiple adversarial clients as they never exchange any information with non adversarial clients and only learn unavoidable information (that they are at risk, if this is the case) from the server. $R\backslash I-IND$ and $I\setminus|I|-IND$ are broken for an adversarial server, as it learns all meeting events and the infection status for all users. This implies that all other notions are broken as well.

Further of course the exact location information is leaked and identities are known to the server.

NTK/ROBERT [125] / [131]

Client. As clients only learn the temporary identifiers (and we ignore trivial linking due to pseudonym reusal during extended periods of time, for the moment) and clients only get notified about whether they are at risk, we expect that one client cannot learn anything about the meetings or who is infected.

Server.[3] The server learns all short-term pseudonyms seen during all encounters of users who test positive, and the long-term pseudonym of the latter. The short-term pseudonyms can be inverted and hence directly mapped to the corresponding users. Linking long-term pseudonyms

[1]Note, that all approaches that were discussed for implementation in Europe assumed short-lived pseudonyms derived from 24h long-term pseudonyms, to reduce communication cost and complexity for deriving at-risk individuals.

[2]The naive solution assumed in the background (Section 7.2.1).

[3]The server is assumed to be trusted in this approach: The NTK/ROBERT description argues that at-risk individuals can be alerted directly by the service, who consequently needs to know who they are. The server thus learns everything about all individuals who test positive and all their encounters, our analysis hence does not match the adversary model of the developers for this part.

(app IDs, cell phone numbers) to the identity of individuals is likely also possible for a strategic adversary, especially with knowledge about the ego-network of the respective victim.

Notions. Corrupted clients only learn the tokens of each batch and the bit about whether they are at-risk (as we consider the fact that multiple users learn that they are meeting the same user due to the broadcast unavoidable leakage, for the moment). As the tokens cannot be linked (again: we ignore the reusage of the pseudonyms for 15 minutes in this analysis) and the bit is considered unavoidable information, $P{-}IND$ is expected to be achieved (and thus also any other defined notion), even against multiple clients in such a restricted analysis.

A corrupted server is expected to break $R{-}IND$ and $I \setminus |I|{-}IND$. As the server notifies the at-risk users and thus learns about their meeting with infected individuals, it learns about the meetings between these users (and most likely even their identities). As the notified honest users can differ in both scenarios for $I \setminus |I|{-}IND$, this notion is also expected to break. Only the notions that allow the adversary to learn about the meetings of infected $R \setminus I{-}IND$ and $M \setminus I{-}IND$ are expected to hold, as the server only learns the short-term pseudonyms of users which the infected users met and nothing about the meetings between non-infected users is shared with the server.

DESIRE [35]

Clients. Clients calculate a specific pseudonym for each meeting partner based on the Diffie-Hellman key exchange protocol. Those pseudonyms are even directional (if Alice and Bob meet, Alice will store a different identifier for the case that Bob gets infected than for the case that she gets infected). Clients can ask the server for subsets of the observed pseudonyms to check whether they are at risk.

Server. The Server learns the pseudonyms of meeting events infected users had. To prevent the server from learning the clients connected to a request a TLS Proxy is proposed.

Notions. Corrupted clients can learn when and how many infected users they met by asking for risk notification of subsets of actually met users. Thereby $M{-}IND, I \setminus |I|{-}IND$ and all stronger notions are broken. Note that the general idea is close to the attack of a user just meeting a small set of other users, achieved e.g. by disabling the communication after meeting the victim. The crucial difference is that in the sketched attack on DESIRE the adversary is able to restrict the set of possibly infected users to be smaller than all users she had meeting events with (as she can simply test subsets of meeting events), which could be prevented. While the second case, the adversary's meeting behavior could as well be an honest user's meeting behavior from someone that just happened to meet no one after meeting the victim. Other than that, we expect corrupted clients to be unable to learn about meetings between benign ($R{-}IND$) and not infected ($M \setminus I{-}IND$) users. Thus $R{-}IND, M \setminus I{-}IND$ are expected to be achieved.

A corrupted server learns how many meeting events infected users had and thereby something about the (meetings of) infected, possibly benign users other than how many infected users exist (breaks $I \setminus |I|{-}IND, M{-}IND, R{-}IND$). However, the server does not learn information about not infected users ($M \setminus I{-}IND, R \setminus I{-}IND$).

ConTraCorona [22]

Clients. Clients use pseudonymous (pids), seed (sids) and warning IDs (wids). Clients send their pids to other clients with the help of secret sharing. One share is sent per minute with a threshold of 15 shares needed to recover the pid. In case of an infection event all observed pids are given to the matching server, which in cooperation with the other servers enables the warning server to retrieve and publish the wids of users at risk.

Server. The tasks of the server are distributed over three different servers: the submission server, the matching server and the warning server. The submission server collects (sid,pid)-pairs from clients and sends them rerandomized and shuffled to the matching server. The matching server also receives the pids that infected users observed. In this event the matching server finds the corresponding sid and sends it to the warning server, that can finally retrieve and publish the wid of the users at risk. For publication the warning server removes duplicates.

Notions. Corrupted clients can learn at which day they met infected users by comparing their wids of the respective day. Thereby, they learn something about meetings and the infected and thus $M{-}IND, I \setminus |I|{-}IND$ and all stronger notions are broken. We would like to stress that the information these clients gain is of the granularity of days, not the exact point in time or smaller time windows like 15 minutes as in other approaches. Yet, it is enough information to break our strong privacy notions. We expect the remaining notions to hold as clients do not learn about when they meet a certain other non-infected user (again) or information about meetings of benign users from the pids and wids they observe.

For a corrupted server, we assume that the for the adversary beneficial server type is corrupted, but only one of the types is. For this, notice that the warning server can count the number of duplicates it has to remove for each wid. Thereby it learns a histogram of how many users had how many meetings with infected on that day; enough information to break our strong notions $R{-}IND$ and $I \setminus |I|{-}IND$. However, none of the servers can learn about meeting events between non-infected users and thereby $M \setminus I{-}IND$ and $R \setminus I{-}IND$ are expected to hold.

Canetti [34]

Clients. Non-infected users only share one-time pseudonyms with other users. We thus expect the meetings of them to be nearly completely private. Infected users send all own temporal identifiers to the server. All users receive the list of pseudonyms of infected users, as it is stored on the server.

Server. The server receives all the identifiers that the infected users sent out.

Notions. $M{-}IND$ and $I \setminus |I|{-}IND$ are expected to break for a single adversarial user, as the user learns in which batch the match with an infected user occurs and thus the user has a good guess which the infected user was (given she knows when she met him: $I \setminus |I|{-}IND$) or when she met him (given she knows when the other user gets infected: $M{-}IND$). We expect $M \setminus I{-}IND$ and $R{-}IND$ to hold against that adversary, as the adversary does not learn about meetings of non-infected users, or meetings she is not involved in. As only the own keys of the infected users are published, it does not contain meeting information.

$I{-}IND$ is expected to break for an adversarial server, as it can observe how many batches of tokens she gets and thus learns the number of infected users. However, given the tokens are sent to the server in a protected way (via a secure anonymous communication network), the server cannot infer the identity from the tokens or any meeting information (as above). We thus expect a corrupted server (not colluding with any clients) to achieve the other notions.

Note that the publishing of *all* tokens of the infected users leaves them traceable by multiple corrupted users, if only one user has been infected in this period (if there have been several, they create an anonymity set). This should be analyzed in future work (see Appendix D.1.2).

MIT [18]

Clients. Clients are assumed to have met if they are in the same region. Clients do not directly exchange information, every communication happens over the central regional server. Infected clients use generalization to obfuscate their location and time, before regularly publishing this information to a database. Non-infected clients run a Private Set Intersection (PSI) or PSI-Cardinality (PSI-CA) protocol together with the database server to find out whether they have been at the same obfuscated location at the same time span as one of the infected clients. We assume that this PSI does not return duplicated set entries (no multi sets). The client learns either the obfuscated location and time for any match in the database (PSI) or the number of matches (PSI-CA).

Server. The server receives regular input from infected users, so it learns the number of infected users. As obfuscated location-time tuples are generated in a deterministic way, infected users that meet in the same generalized location at the same time will generate the same information to be sent to the server. The server thus learns a histogram over the number of infected people over time that meet at the same location. Depending on the chosen PSI or PSI-CA scheme, the server also can (on its own, or together with a colluding client[4]) perform a bruteforce attack on the time span and obfuscated location reported by any infected user.

Notions. Note that the hashed obfuscated location-time value stored at the server and received by the clients can be efficiently resolved to the obfuscated location and time by a dictionary first preimage attack (in polynomial time in the security parameter).

$M-IND$ and $I \setminus |I|-IND$ are expected to break for corrupted clients as they can learn in which batch they had contact to a corrupted user (as above for Canetti). The other (not yet implied to be broken) notions are expected to hold for this adversary as corrupted users learn no meeting information about users they did not meet or that are not infected, as they can only check for their regional information against regions infected have been in.

$R-IND$ and $I-IND$ are expected to break for an adversarial server, as he learns the (obfuscated) location and time of infected users and can reconstruct a partly social graph and number of infected users. Further, $I \setminus |I|-IND$ is expected to break as the server can learn about two infected users that they have met (been at the same region at the same time) and thus can construct a scenario, where these met and another where the two infected (different users than in the first scenario) did not meet, even though in both the same number of users is infected and the same users (identities) met. We expect $M \setminus I-IND$ to hold, as the server does not get information about the meetings of non-infected users (if the PSI is secure).

Note that the description of possibly achieved notions does not leverage the location stored in the obfuscated location-time tuple. This is due to our assumption that location of adversarial users is identical in the same batch for the moment and should be analyzed in future work.

DP3T [152]

Note that the different versions of DP3T (low-cost, unlinkable and hybrid) vary the linkability of pseudonyms. For this first analysis, we only informally state the expected effect of the pseudonyms in Table D.2, but otherwise argue the notions independent of pseudonym linkability.

Further, while the different versions provide different protection measures, in terms of our first selection of strong notions, they all achieve the same subset, which is why we discuss only the low-cost protocol in the paragraphs below and quickly mention why the differences do not influence our notion selection here. Relaxed and fine-grained notions that indeed reflect the differences are however an interesting direction for future work.

[4]He can simply take part in the open world app as a user.

The unlinkable and hybrid version allow users to redact pseudonyms from certain time spans. If we assume that not all meetings (with corrupted clients) are redacted, this does not influence the analysis of our strict notions, as information about the non-redacted meetings can still be used in the attack. Further, in all versions the client gets to check for the warning based on the received tokens, she thus of course learns the alert producing token and thereby the round in which the contact happened, which is the major idea to break our notions as detailed below.

Clients. Clients continuously broadcast a cryptographic token via bluetooth. This token is derived from a temporary key. The key is supposed to have a validity of roughly one day. The next key is derived from the current temporal key. Each temporal key generates n tokens, which are used one after another over the lifetime of the temporal key. Clients receiving such tokens are storing them together with a timestamp and signal strength. In case of an infection, the app of the infected user sends the temporal key together with a validity timespan to the server. Once a day, each client receives the server list of keys and their validity. It generates all belonging tokens and possibly finds matches with recorded tokens.

Server. The server receives a tuple consisting of a temporal key and its validity in case of an infection. Clients request the list of tuples once a day.

Notions. As above (for Canetti and MIT), $M{-}IND$ and also $I \setminus |I|{-}IND$ are expected to be broken for a corrupted client, as she learns in which round she met an infected user (she calculates the matches on her own). However, we expect the other (not implied to be broken) notions to hold, as again nothing about the non-infected meetings, or meetings that do not involve a corrupted user, can be learned.

Similar to the case with Canetti, an adversarial server learns the number of infected users (from the number of keys it gets sent) and thereby breaks $I{-}IND$. However, other than that the server only learns the keys of infected, but no further meeting information and thus we expect the other (not implied to be broken) notions to hold.

Note that, as acknowledged by the authors, the publishing of keys of the infected users in the low-cost version allows to derive all sent tokens of this user and leaves her traceable by multiple corrupted users similar to the Canetti approach.

Remarks

We consider some observed pseudonyms to be unavoidable leakage, which is only due to the fact that we assume the tokens to be transmitted via a broadcast medium (which all promising contestants for a privacy-preserving solution in Europe currently are suggesting, using 128-bit BLE tokens). It fundamentally is possible to share them via unicast in an encrypted fashion. Bystanders then cannot observe them, and each pseudonym relates to only the tuple of two users that participate in the concrete encounter. This characteristic was illustrated e.g. with DESIRE.

Further, we have repeatedly sketched attacks against our strong notions by exploiting that a user learned the fact *that* and *when* she met an infected user. With the current functionality and the goal to calculate the warning at the client this seems challenging to avoid. An easier circumvention of this problem is to allow for added delays and collect information to infer warnings for this time. The concrete analysis of this however needs an adapted notion that forces the adversary to include multiple infection events in this time span.

D.2. Payments

D.2.1 Hierarchy Proof Sketches

Sender Notions (Receiver similarly). Except (Sender-Value) Unlinkability, Sender Unlinkability Fixed Value, Sender Unlinkability Fixed Total, Sender Unobservability Fixed Value and Sender Unobservability leaking Graph the notions are defined as the sender notions in anonymous communication networks and the proofs follow similar to their equivalent relation in anonymous communication. The *Atom* property is the only difference for (Sender-Value) Unlinkability, but only weakens the notion more than without and hence the proofs still apply.

Sender Unlinkability leaking Pseudonym ...

- ... implies Sender Unlinkability Fixed Value: If Q and F_V, P has to contain only multisets of the fixed value and the number of times the fixed value is in each multiset is the same due to Q.

Sender Unlinkability Fixed Value ...

- ... implies Sender Unlinkability Fixed Total: trivial.

Sender Unlinkability Fixed Total ...

- ... implies 2-(Sender-Receiver) Unlinkability: Due to *Atom* we can reorder the two payments such that only receivers differ in each row. The values for both payments are the same and each sender sends once. Every choice for 2-(Sender-Receiver) Unlinkability is thus also valid for Sender Unlinkability Fixed Total.

Sender Unobservability ...

- ... implies not Value Unobservability leaking Graph: A protocol that publishes all payed values (e.g. stores them on the blockchain) but hides everything else perfectly, has Sender Unobservability, but no Value Unobservability leaking Graph.

Sender Unobservability leaking User Number ...

- .. implies not Sender Unobservability leaking Graph: Assume a protocol that leaks the number of active users, but hides everything else perfectly. It has Sender Unobservability leaking User Number, but not Sender Unobservability leaking Graph.

Sender Unobservability leaking Graph...

- ... implies not 2-(Sender-Receiver) Unlinkability, Twice Sender Unobservability or (Sender-Value) Unlinkability: A protocol that leaks the link weights of the credit/payment network, but hides everything else perfectly, has Sender Unobservability leaking Graph, but not 2-(Sender-Receiver) Unlinkability, Twice Sender Unobservability and (Sender-Value) Unlinkability.

Twice Sender Unobservability...

- ... implies not Sender Unobservability leaking Graph: A protocol that leaks the sender of the first transaction, but hides everything else perfectly, has Twice Sender Unobservability, but not Sender Unobservability leaking Graph.

The receiver notions follow similarly.

Impartial Notions. We discuss implications starting at an impartial notions here.
Unobservability...

Table D.3: Example attack choice

Sender	Value	Receiver	Sender	Value	Receiver
A	1	R	A	2	R
B	2	R	B	1	R

Table D.4: Example: Smaller than max balancing payments after middle line

Sender	Value	Receiver	Sender	Value	Receiver
A	2	R	A	1	R
B	1	R	B	2	R
A	0.5	R	A	1.5	R
B	1.5	R	B	0.5	R

- ... implies Value Unobservability, Receiver Unobservability, Sender Unobservability: trivial

Value Unobservability ...:

- ... implies Value Unobservability leaking Graph: trivial.
- .... implies (Sender-Value) Unlinkability, (Receiver-Value) Unlinkability: similarly to the anonymous communication case, except that (Sender-Value) Unlinkability has even more restrictions (i.e. is even weaker) than the equivalent in the anonymous communication case.
- Remaining counterexamples follow from Unobservability not implying the other notions.

Value Unobservability leaking Graph ...:

- ... implies not (Sender-Value) Unlinkability and (Receiver-Value) Unlinkability: Assume a protocol that only leaks the graph. This achieves Value Unobservability leaking Graph as there $\equiv_G$ is required. However, the graph leaks who sent which value as long as no balancing payments occur, thus breaking (Sender-Value) Unlinkability, e.g. with the choice from Table D.3.

Unobservability Fixed Value...:

- ...implies Unlinkability Fixed Value: trivial.
- ...implies not (Sender-Value) Unlinkability: Assume a protocol that only leaks the value of the first payment of the lexicographically first sender. As this will always be v for any game in Unobservability Fixed Value (due to F_V), this protocol achieves Unobservability Fixed Value. However, the leaked linking allows to break (Sender-Value) Unlinkability, e.g. with the choice from Table D.3.
- ...implies not Sender Unlinkability leaking Pseudonym: Same example protocol and attack example as for (Sender-Value) Unlinkability.
- ...implies not Twice Sender Unobservability: Assume a protocol that only leaks whether two consecutive transactions are from the same sender if their values differ. As the values will always be the same in Unobservability Fixed Value (due to F_V), Unobservability Fixed Value is achieved. However, learning if someone sent twice allows to break Twice Sender Unobservability.
- ... implies not Value Unobservability leaking Graph: Assume a protocol that only leaks the maximal value of any payment of the lexicographically first sender. As this will always be v for the Unobservability Fixed Value game (due to F_V), this protocol achieves Unobservability Fixed Value. However, the leaked linking allows to break Value Unobservability leaking Graph, as the balancing can be made with smaller payments (see Table D.4).

Unlinkability Fixed Value...:

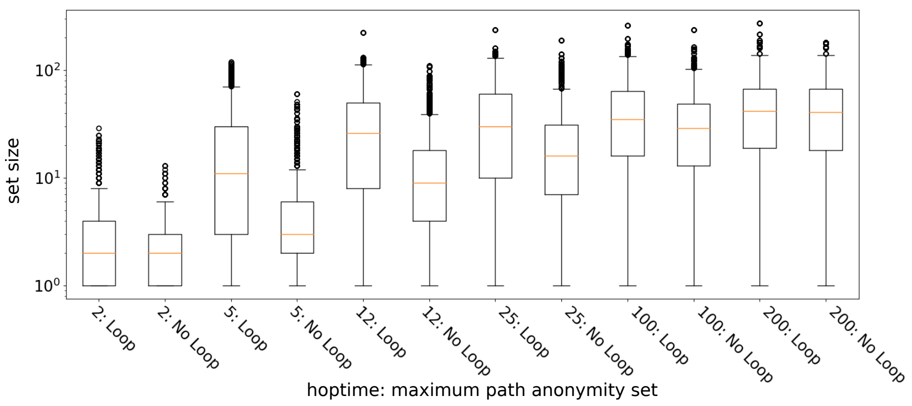


Figure D.1: Effects of allowing loops illustrated with the maximum path anonymity set sizes of varying hoptime for the scaled buckets. The differences in the anonymity set sizes caused by allowing loops are smaller for very short or long hop times. However the increase when including loops is never high enough to justify introducing loops to the routing as improvement mechanism.

- ...implies Unlinkability Fixed Total, Sender Unlinkability Fixed Value, Receiver Unlinkability Fixed Value: trivial.

Unlinkability Fixed Total...:

- ... implies Sender Unlinkability Fixed Total, Receiver Unlinkability Fixed Totals: trivial

D.2.2 The Effect of Loops

So far we assumed a careful adversary that knows the routing strategy and ignores paths with loops for the maximum anonymity sets. However, to see how much influence loops have on the sets, we calculated the maximum anonymity sets that also allow for loops on the path and show the difference in Figure D.1. For short hop times the difference is not high as generally not many payments are mixed and hence not many loops are constructed transitively. For very large hop times, the loops also do not change the anonymity set much as there are not many different time slots in which effects of the transitivity could occur. However, for intermediately long hop times a difference can be seen. As this however not only includes paths with one, but also multiple loops and as allowing for loops in the paths supports congestion attacks [62], we do not deem including loops to the paths as a useful countermeasure, but suggest increased hop times instead.

D.2.3 Relation to Protocols

We want to illustrate our notions by setting them in relation with prominent P3 systems. For this we select P3 systems that have the most precise goal definitions or have been very impactful. Our first broad comparison is limited in scope: we match our notions to the goals that *the proposals state*[5] for their chosen adversary model, as we subsequently want to investigate their relationship with regards to anonymity set sizes for a selection of the protocols. We therefore do not consider the proofs, potentially differing adversary models, or assumptions.

This has, first of all, the effect that a protocol might appear weaker than another even though they are not directly comparable: The weakness might stem from the chosen setting, adversary or desired functionality. If, for example, the transmitted value is needed to calculate the fees, the protocol *has to* leak the values to some participants. A protocol thus might achieve a weaker notion only, just because of the desired functionality.

[5]We do not take any additional attack vectors (such as information leaked on the network layer and user behavior heuristics [84]) into account, but just work with the claims of the systems.

Secondly, to allow for a broad comparison, we only focus on the goals that the P3 systems state (formally or informally). We thus ignore sophisticated attacks like traceability analyses for Monero [101, 118] or side channel attacks [151].

Table D.5 shows the relation between the privacy goals of P3 systems and our notions. We discuss the mapping below.

Table D.5: Relation between Notions and Protocol Claims

Protocol	**Notion**	**Remark**
Chaum's eCash	Unobservability Fixed Value	all
CoinShuffle++	Sender Unobservability	only for participating peers
Fulgor + Rayo	Value Unobservability 2-(Sender-Receiver) Unlinkability	all, common intermediate node & plausible paths
Monero	Sender Unobservability Receiver Unobservability Value Unobservability	only for subset chosen by sender all all
PathShuffle	Sender Unobservability	only for participating peers
PrivPay	Value Unobservability leaking Graph Receiver Unobservability leaking Graph	all, all
Quisquis	Unobservability	only for subset chosen by sender
SilentWhispers	Value Unobservability leaking Graph Sender Unobservability leaking Graph Receiver Unobservability leaking Graph	all, all, all
TumbleBit (payment hub) TumbleBit (classic tumbler)	Unlinkability Fixed Value Unlinkability Fixed Total	all, all
Zerocash	Unobservability	all
(Anonymous) Zether	Unobservability	only for subset chosen by sender

Mapping Formal Goal Definitions

CoinShuffle++. *CoinShuffle++* [137] applies a DC-net approach from anonymous communication to the payment setting to achieve sender anonymity.

CoinShuffle++ requires *Sender Anonymity* for successful runs. For sender anonymity each real honest sender should be indistinguishable from all other honest peers. Thus scenarios can differ arbitrarily in the senders (E_S) and should still be indistinguishable, corresponding to our Sender Unobservability.

Fulgor + Rayo. *Fulgor and Rayo* [109] are two privacy-enhancing approaches for PCNs.[6] In PCNs participants lock funds on the blockchain via smart contracts to then be able to do multiple payments before closing the channel and updating the blockchain accordingly. Both, Fulgor and Rayo, aim at hiding both, the value of payments from nodes that are not on the path, as well as the relationship between sender and receiver even against nodes that are on the payment's path. Fulgor and Rayo differ however in their treatment of concurrent payments and in the adversary model against which all their privacy goals are achieved, which are both out of scope for this work.

Fulgor and Rayo describe the two privacy goals *(Off-path) Value Privacy* and *(On-path) Relationship Anonymity.* (Off-path) Value Privacy states that no outsider (not involved in the current

[6]While there are additional interesting approaches, such as Bolt [76], we choose Fulgor and Rayo for our analysis due to their compatibility to Bitcoin and easy extensibility to the Lightning network.

payment path) learns information about the value. As our notions remove the adversary from the privacy goal[7], we model this as allowing the value to differ arbitrarily between the two scenarios (E_V) resulting in Value Unobservability.

(On-path) Relationship Anonymity states that for any two simultaneous payments with a common honest intermediate node on their path, the other nodes cannot determine the sender-receiver combination for any of these payments. We dissect this definition in the basic properties *Atom* (payments are simultaneous), $|P|_2$ (exactly 2 payments are involved) and M_{SR} (the sender-receiver linking is the only thing that differs and the corresponding paths are included in the auxiliary information). This results in 2-(Sender-Receiver) Unlinkability.

PrivPay. *PrivPay* [114] uses oblivious capacity calculations supported by trusted hardware at its core to protect the credit links, as well as value and receivers of payments.

PrivPay aims at *value privacy* and *receiver privacy* for payments with honest senders. Value privacy is defined as a game, that only allows the values to differ, but at the same time adds balancing transactions that ensure the same resulting credits on the credit network links for both scenarios. This excludes trivial wins for the adversary. This notion corresponds to our Value Unobservability leaking Graph, where E_V ensures that only values can differ, $\equiv_G$ ensures the equivalent credit networks and *Atom* that no intermediate credit network states are output to the game adversary. Note that PrivPay's balancing transactions that ensure equivalent graphs are implicit in our $\equiv_G$ requirement.

Receiver privacy for payments with honest senders, translates to our Receiver Unobservability leaking Graph that allows only receivers to differ in the scenarios (E_R) and ensures an equivalent credit network through balancing transactions ($\equiv_G$). We however decouple this goal from the adversary model dimension – requiring that the sender is honest is not included in our notion, as we consider this to be part of the adversary model.

SilentWhispers. *SilentWhispers* [110] aims at keeping not only the value as well as the sender and receiver of a transaction hidden, but also the credits on the links. The authors suggest to employ a secret-sharing multiparty-computation combined with signatures to reach payment integrity and these privacy goals.

SilentWhispers requires *value privacy*, *link privacy*, *sender privacy* and *receiver privacy*. Value privacy and receiver privacy are defined as in PrivPay and hence map to the same notions. Further, sender privacy is defined similar to receiver privacy and thus results in our corresponding sender notion Sender Unobservability leaking Graph. Link privacy requires that the adversary cannot learn the available credit between two honest users. As this is not directly related to any payment, we consider this kind of leakage out of scope for this paper.

TumbleBit (payment hub). In *TumbleBit* [81] it is the task of a Tumbler, an untrusted intermediary, to mix the payments of all senders using the Tumbler at the same time under each other, such that it is hidden which receiver was payed by which sender. Cryptographic puzzles ensure the correctness and privacy of the TumbleBit approach.

As a payment hub, TumbleBit aims at *Unlinkability* which is defined based on the view of the (adversarial) tumbler on the interaction graph. The interaction graph is a representation for the mapping of payments to senders and receivers. The view of the tumbler is limited to the puzzle-solver protocols with each sender. Thus, the tumbler learns the senders and how many payments they make, and obeserves the cashout phase, i.e. the tumbler learns how much each receiver gets payed. Unlinkability requires that all interaction graphs that induce the same view for the tumbler are equally likely.

[7]This means in the analysis the *protocol model* cannot return outputs that correspond to on-path nodes, but our notion is suitable.

Translated to our model the interaction graphs are the single scenarios and as all of them have to be equally likely, we know that each such pair, and especially the worst case pair for the protocol, has to be indistinguishable. This is thus equivalent to our indistinguishability game, restricted by properties that correspond to the tumbler's view. Before translating the tumbler's view, we note that TumbleBit's setting assumes that all payments are of the same denomination, hence we use F_V in the corresponding notions. The tumbler's view includes the number of payments per sender (Q_S). This in combination with the fixed values ensures also equal totals for the senders (Σ_S). On the receiver side the tumbler learns the total (Σ_R) which together with the fixed values also ensures equal receiving frequencies (Q_R). This corresponds to our notion Unlinkability Fixed Value (as we do not need auxiliary information to model TumbleBit, requiring E_{SR} additionally makes no difference).

TumbleBit (classic Tumbler). As a classic Tumbler, TumbleBit aims at *k-Anonymity* which is described as revealing which senders and receivers participated in an epoch, but not being able to link senders to receivers for all k successful payments. Thus, in this setting the values are fixed to 1 and each participant either sends or receives 1 coin. This ensures that the totals are not only equal, but fixed to 1 as well ($F_{\Sigma_S \Sigma_R}$). The resulting goal corresponds to our Unlinkability Fixed Total. While the authors argue that this second property is stronger because senders and receivers can no longer be grouped based on the totals they send/receive, these larger anonymity sets stem from the usage assumption (everyone sends/receives exactly one coin), not from hiding additional metadata. Indeed from the metadata point of view the first notion is stronger.

Quisquis. *Quisquis* [63] suggests using updatable keys and ZK arguments to add deniability of participation to the anonymity requirements.

Quisquis defines *Anonymity* both for senders and receivers in an anonymity game. While their detailed game mixes parts of the adversary model, like corrupting secret keys for transactions, to the goal of indistinguishability of challenge transactions, the distinguishing feature of Quisquis's game definition is the inclusion of the anonymity set as part of the transaction. Other than that only the same number of transactions in both scenarios follows from their query definition (which is equivalent to E_{SRV} without using auxiliary information), which terminally corresponds to our notion Unobservability.

Zerocash. Zerocash proves to achieve *Ledger Indistinguishability*, defined as:

> "First, a challenger samples a random bit b and initializes two DAP scheme oracles $\mathcal{O}_0^{DAP}$ and $\mathcal{O}_1^{DAP}$, maintaining ledgers L_0 and L_1. Throughout, the challenger allows $\mathcal{A}$ to issue queries to $\mathcal{O}_0^{DAP}$ and $\mathcal{O}_1^{DAP}$, thus controlling the behavior of honest parties on L_0 and L_1. The challenger provides the adversary with the view of both ledgers, but in randomized order: $L_{Left} := L_b$ and $L_{Right} := L_{1-b}$. The adversary's goal is to distinguish whether the view he sees corresponds to $(L_{Left}, L_{Right}) = (L_0, L_1)$, i.e. $b = 0$, or to $(L_{Left}, L_{Right}) = (L_1, L_0)$, i.e. $b = 1$. At each round of the experiment, the adversary issues queries in pairs Q, Q' of matching query type. If the query type is CreateAddress, then the same address is generated at both oracles. If it is to Mint, Pour or Receive, then Q is forwarded to L_0 and Q' to L_1; for Insert queries, query Q is forwarded to L_{Left} and Q' is forwarded to L_{Right}. The adversary's queries are restricted in the sense that they must maintain the public consistency of the two ledgers. For example, the public values for Pour queries must be the same, as well as minted amounts for Mint queries."

Zerocash's *Ledger Indistinguishability* includes an informal part that restricts the scenario to "maintain the public consistency of the two ledgers" without defining what needs to be equal for the public consistency. Further, this game *always* outputs the observations to both scenarios.

While theoretically giving the observations to both scenarios is stronger than the classical definition that returns just one, we are not aware of any practical relevance of this decision. Hence, we decide to map this to a notion of our hierarchy, even though the models differ in this way.

As we are interested in the transactions, we focus on the restrictions for Pour and Receive queries due to the public consistency requirement. If we assume that all parameters important for the public consistency, like the public values for the pour queries, are specified in the auxiliary information and do not restrict the other sender, receiver or value choices, we can map Zerocash's goal to *Unobservability.*

(Anonymous) Zether. *Zether* [29] uses ZK proofs to improve, besides other possible applications, Ethereum's privacy by adding Zether smart contracts.

Zether defines all privacy requirements in a *Privacy-Game.* While the game allows further queries, like to lock and unlock accounts, for the transfer transaction the only requirements are that the senders have sufficient funds, that the same anonymity set is used in both scenarios and if the receiver is corrupt, the same receiver and amount are used. We assume the first two always as well and the third one corresponds to the user corruption that can be similarly added to our notions as in the anonymous communication case (see Appendix A.1.2). We again exclude empty payments via E_{SRV}, which leads to Unobservability.

Mapping informal Claims of Protocols

Note that the protocols discussed in this section do not have easily comparable goal definitions. Rather, their goals are informal. We hence stress that this is our interpretation of the informal definitions and that an extensive, in-depth analysis regarding all notions is required for each of the P3 systems, to confirm or reject our intuition in future work.

Monero. Monero claims to be *confidential and untraceable* by hiding the senders with ring signatures, the receivers with stealth addresses and values with ring confidential transactions. It however does not define any of these three protection goals in detail. For senders, we need to reduce the anonymity set to the users corresponding to the ring signature, but do not have to restrict the notion otherwise. For receivers and values, the protection's claim includes all other existing receivers (and values) into the anonymity set and causes no further restrictions. We thus map them to Receiver Unobservability and Value Unobservability.

Chaum's eCash. *Chaum's eCash* [39] describes a payment system that improves privacy based on blind signatures.

eCash's goal is the "inability of third parties to determine payee, time or amount of payments made by an individual". This allows to leak the total number of payments and as eCash assumes fixed values, also the value of each single transaction, but hides everything else. This informal goal and the setting of eCash hence match most closely to our Unobservability Fixed Value.

PathShuffle. *PathShuffle* [115] proposes a path-based mixing protocol by realizing atomic transactions between multiple users. Thereby they realize unlinkability between output wallet and user.

PathShuffle defines its privacy goal "Unlinkability" informally as the fact that "it should not be possible for the attacker to determine which output wallet belongs to which honest user". More precisely we expect that they mean for all honest users *participating in this protocol run.* Other than that it expresses the uncertainty of the adversary in which user (sender) is correlated to which output wallet (receiver and transaction). Hence this matches Sender Unobservability most closely.

Relating Simulation Results

With our mapping of protocols to notions from Section D.2.3, we can also relate them to our anonymity set simulation. However we would like to remind the reader that the simulation represents a simplified overview and abstracts from details of the different systems, as for instance the actual epoch or hop time or changes in the set of all users due to users joining and leaving the P3 system (churn).

The anonymity set of *Zerocash* corresponds to the set of all users (Sender Unobservability without anonymity set restriction, i.e. $\widetilde{\mathcal{U}} = \mathcal{U}$). This is the biggest possible sender anonymity set, as the protocol protects even the fact if users are currently active, or not.

While the goal definition in PathShuffle and CoinShuffle++ is given similar to Sender Unobservability the additional restriction ensures that their goal actually corresponds to Sender Frequency Unlinkability. Thus it corresponds to *the set of active users*. Notice the importance of the epoch time for the corresponding anonymity set sizes. Naturally, we expect an epoch to be a limited for the mixing in PathShuffle and CoinShuffle++, but the longer it is, the bigger is the anonymity set sizes are.

The anonymity sets of *TumbleBit* are best approximated by *the active senders with identical values* (Sender Unlinkability Fixed Value, the sender notion implied by Unlinkability Fixed Value), the group of the smallest anonymity sets in the experiments above without considering path-based restrictions.

There is one more category of systems achieving the strongest notion. The approaches *Monero*, *Quisquis* and *Zether* achieve Sender Unobservability, but explicitly limit the anonymity set and (assuming none of the users in it is corrupted and the selection is not biased) ensure a fixed anonymity set size. The sizes are still being discussed and subject to change. E.g. *Monero* uses 11 and for *Quisquis* 4, 16 or 64 other senders are picked in their evaluation.

The evaluation of path-based approaches finally corresponds to PCNs like *Fulgor* and *Rayo*. Their anonymity set sizes are best approximated by our evaluation considering *path overlap and value identity.*

Take away messages. Depending on the chosen parameters (concrete hop or epoch time, as well as group sizes for Monero, Quisquis and Zether) the active, value or even path-based anonymity sets of TumbleBit, PathShuffle or Fulgor and Rayo might perform better or worse than the fixed values of Monero, Quisquis and Zether. Their expected anonymity set sizes at least can be on a similar order of magnitude, when compared to the protocol choice of the latter. Although CoinShuffle++ achieves the strongest notion, the overall expected anonymity sets of active users for Zerocash reach even better sizes. The reason is the difference in the additional assumption.

Limitations. Recall that our protocol assessment has to be taken with grains of salt. We compared systems built on different fundamental assumptions, only with regards to the dimension of privacy. Each design decision obviously entails differences in performance, like in terms of latency, bandwidth, computational complexity and scalability, which we have not considered at all in this work.