Abstract of "Onions, Shallots and Leaks:
Anonymous Communications Through Public Networks"
– by Megumi Ando, Ph.D., Brown University, May 2020.

The downside of the world becoming more digitally connected is that it is now easier for powerful adversaries to strongly discourage digital communications by conducting mass surveillance. This has a chilling effect on freedom of speech. While proper encryption and authentication can secure the confidentiality of messages' content, communication patterns (i.e., who is communicating with whom) can still leak from observed traffic over the Internet. Onion routing is the most promising method for enabling anonymous communications. In an onion routing protocol, messages travel through several intermediaries before arriving at their destinations; they are wrapped in layers of encryption (hence they are called "onions"). Despite the widespread use of onion routing in the real world (e.g., Tor, Mixminion), the foundations of onion routing have not been thoroughly studied. In this dissertation, we present new results on onion routing protocols and onion encryption schemes that lend themselves to onion routing. Our results include:

1. An anonymous protocol with polylogarithmic (in the security parameter) onion cost (onions transmitted per party) in the presence of the passive adversary who can monitor a constant fraction of the nodes.

2. A differentially private protocol with polylogarithmic (in the security parameter) onion cost in the presence of the active adversary who can control a constant fraction of the nodes.

3. The first onion routing protocol that is simultaneously efficient, fault-tolerant and anonymous in the active adversary setting. The protocol requires an expected polylogarithmic (in the security parameter) number of onions to be transmitted per message.

4. A lower bound (matching our protocol) for achieving anonymity in the active adversary setting. We show that for an onion routing protocol to be anonymous and fault-tolerant, the onion cost is $\omega(\log \lambda)$, where λ is the security parameter.

5. We also present the first provably secure "repliable" onion encryption scheme for enabling the recipient of an onion to reply to the anonymous sender. Our work resolves the previously open problem of formalizing onion encryption for two-way channels.

Abstract of "Onions, Shallots and Leaks:
Anonymous Communications Through Public Networks"
– by Megumi Ando, Ph.D., Brown University, May 2020.

The downside of the world becoming more digitally connected is that it is now easier for powerful adversaries to strongly discourage digital communications by conducting mass surveillance. This has a chilling effect on freedom of speech. While proper encryption and authentication can secure the confidentiality of messages' content, communication patterns (i.e., who is communicating with whom) can still leak from observed traffic over the Internet. Onion routing is the most promising method for enabling anonymous communications. In an onion routing protocol, messages travel through several intermediaries before arriving at their destinations; they are wrapped in layers of encryption (hence they are called "onions"). Despite the widespread use of onion routing in the real world (e.g., Tor, Mixminion), the foundations of onion routing have not been thoroughly studied. In this dissertation, we present new results on onion routing protocols and onion encryption schemes that lend themselves to onion routing. Our results include:

1. An anonymous protocol with polylogarithmic (in the security parameter) onion cost (onions transmitted per party) in the presence of the passive adversary who can monitor a constant fraction of the nodes.

2. A differentially private protocol with polylogarithmic (in the security parameter) onion cost in the presence of the active adversary who can control a constant fraction of the nodes.

3. The first onion routing protocol that is simultaneously efficient, fault-tolerant and anonymous in the active adversary setting. The protocol requires an expected polylogarithmic (in the security parameter) number of onions to be transmitted per message.

4. A lower bound (matching our protocol) for achieving anonymity in the active adversary setting. We show that for an onion routing protocol to be anonymous and fault-tolerant, the onion cost is $\omega(\log \lambda)$, where λ is the security parameter.

5. We also present the first provably secure "repliable" onion encryption scheme for enabling the recipient of an onion to reply to the anonymous sender. Our work resolves the previously open problem of formalizing onion encryption for two-way channels.

Onions, Shallots and Leaks:
Anonymous Communications Through Public Networks

by

Megumi Ando
B.S. in Mathematics, M.I.T., 2007
B.S. in Electrical Engineering and Computer Science, M.I.T., 2010
M.Eng. in Electrical Engineering and Computer Science, M.I.T., 2010

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2020

This dissertation by Megumi Ando is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Anna Lysyanskaya, Director

Date _____ _____

Eli Upfal, Director

Recommended to the Graduate Council

Date _____ _____

Leonid Reyzin, Reader
(Boston University)

Date _____ _____

Roberto Tamassia, Reader

Approved by the Graduate Council

Date _____ _____

Andrew G. Campbell
Dean of the Graduate School

To my family.

Acknowledgements

Foremost, I thank my two advisors, Anna Lysyanskaya and Eli Upfal, for many things! Firstly, I thank them for training me. Learning how to do research was my main objective for going back to school, and I feel that I have gained the experience I sought! It was exciting to pioneer together the new and interdisciplinary area of theoretical anonymity, relying on my advisors' different backgrounds. I thank Anna for admitting me to the program and for believing in my potential first. I thank her also for teaching me how to be rigorous in my definitions, notation and, ultimately, my thinking. I thank Eli for admitting Lorenzo (my husband) to the program since, otherwise, I may not have met him in a timely fashion! I also thank him for shaping my perspective as a theorist. I have a much better understanding of what makes for beautiful and elegant theory as a result. I also thank my thesis committee members, Leo Reyzin and Roberto Tamassia, for their technical feedback and valuable career advice!

Being in two research groups has meant that I had the pleasure of having many academic siblings. I thank Ahmad Mahmoody, Liz Crites, Apoorvaa Deshpande, Lorenzo De Stefani, Cyrus Cousins, Leah Rosenbloom, Benedetto Buratti, Alessio Mazzetto, Miranda Christ and Conrad Zborowski for their camaraderie over the years. Special thanks go to Liz for helping me edit my English in places. Grad school can be tough at times, and it was especially helpful to have as moral support, honorary academic siblings, Evgenios Kornaropoulos and Marilyn George. Special thanks go to Marilyn for feeding me when Lorenzo was out of town; she saved me from an "instant ramen only" diet on more than one occasion.

I want to also acknowledge the support I received from my friends at The MITRE Corporation: Chris Alicea, Joshua D. Guttman, Jeff Picciotto, J̈ohn D. Ramsdell, Gabby Raymond and Paul D. Rowe. I thank friends, Bao, Horacio, Jenny, Leonard, Mark, Machiko and Mike for being patient with me for being non-existent these past few years and for rooting me on regardless.

Last but not least, I thank my family. My big sister, Nozomi, who has been my number

one ally in all things life-related for as far back as I can remember. I thank my brother-in-law, Buz, for discussions on everything from science to politics to toilet humor. I thank my Italian parents, Ivana and Antonio, firstly, for raising Lorenzo and also for their constant positivity and understanding. My bunnies, Uchan and Fundo, have been the best companions for thirteen-plus years. I thank my parents, Teiichi and Sumiko, for their love and encouragement and for putting up with me during my terrible twenties. Finally, I thank my husband, Lorenzo for everything. He has been (and will always be) my best friend and greatest cheerleader in all my endeavors.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

To meet today's standards of convenience, the world is becoming more reliant on complex and non-homogeneous systems involving hardware, software, networks, data and people (e.g., cloud computing, IoT, automated cars and the smart grid).

While the "automation of things" can have positive societal and economical impacts, an increasingly interconnected world threatens our fundamental right to privacy. Automated systems store, communicate and compute on private data, which, without taking extra precautions, can leak unwanted private information. Thus, a general goal is to design future systems with minimal private information leakages.

While proper encryption and authentication can secure the confidentiality of messages, communication patterns (i.e., who is communicating with whom) can still leak from observed traffic in a distributed system. In this dissertation, we present new results on routing protocols and encryption schemes for suppressing such leakages.

1.1 Onion routing (OR) protocols

Consider the following example scenarios:

- In the first scenario, Anna is an employee of an authoritarian government. She wishes to expose the wrongdoings of the government by sending her story to the NYTimes without being exposed as the whistleblower.
- In the second scenario, Anna is a regular citizen in a country dictated by an authoritarian government. She wishes to access certain banned websites and applications (e.g., the NYTimes, Wikipedia, DuckDuckGo, Slack) without Internet traffic revealing her browsing history.

- In the third scenario, Field Agent Anna wishes to send a covert message to Field Agent Roberto so that no one, including (possibly double agent) Roberto can determine her IP location. Her message may traverse an autonomous system (AS) largely controlled by a powerful country.

As illustrated by these examples, the downside of the world becoming more digitally connected is that it is now easier for powerful adversaries to strongly discourage digital communications by conducting mass surveillance of individuals' Internet activities.

Anonymous channels are a prerequisite for protecting user privacy. But how can we achieve anonymous channels in an Internet-like network that consists of point-to-point links? Suppose that Anna wishes to send a message anonymously to Roberto. To begin with, Anna can encrypt the message and send the encrypted message to Roberto so that only Roberto can read the message. However, an eavesdropper, Eli, observing the sequence of bits coming out of Anna's computer and the sequence of bits going into Roberto's computer can still determine that Anna and Roberto are communicating with each other if the sequences of bits match. So, Anna can use onion routing to send her message to Roberto instead.

Onion routing [Cha81] is the most promising approach to anonymous channels to-date. In onion routing, messages are sent via intermediaries, and wrapped in layers of encryption, resulting in so-called onions; each intermediary's task is to "peel off" a layer of encryption and send the resulting onion to the next intermediary or its final destination. The onion "looks different" at every layer, and so its route through the network cannot be traced from merely observing the sequences of bits that Anna transmits and Roberto receives. However, even with Anna sending her message to Roberto encoded as an onion, her communication can still be tracked by a resourceful eavesdropper with an extensive view of the network traffic (e.g., an ISP-level or an AS-level adversary) who can observe the "flow" of Internet traffic.

Tor (which stands for "The onion router") [DMS04] is a widely-used distributed network consisting of thousands of relay nodes and used by two million users per day[1] to communicate anonymously[2]. Tor boasts efficiency, *fault tolerance* (i.e., if an onion gets dropped, other onions still make it through) and *scalability* (i.e., the system continues to perform well even as more parties join). These practical benefits have so far outweighed concerns that onion routing lacks provable security in the standard setting where the adversary has

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

[2] Tor is somewhat different than the onion routing protocol as described above, but we begin with the simpler version of onion routing.

the full view of the network traffic [JWJ+13, SEV+15, WSJ+18]. Johnson et al. [JWJ+13] showed that approximately 80% of users can be deanonymized by an adversarial relay node within six months, and almost all users can be deanonymized by AS within three months. This is detrimental when combined with the fact that one-third of all Tor relay nodes are hosted by only six autonomous systems [SEV+15][3].

In this thesis, we explore how we can achieve provable security from powerful (e.g., AS-level) adversaries without completely sacrificing the practical properties we already enjoy in Tor: efficiency[4], fault tolerance and scalability.

Adversary model For our results on onion routing, we will assume that the adversary can access network traffic on all links, but that message packets are dropped only at corrupted nodes. When a message packet is sent directly from Anna to location Leo[5], in actuality, the packet is routed through many physical nodes in the network; thus, it is reasonable to assume that the message could be logged somewhere en route. An adversary controlling a large AS is well positioned to eavesdrop on the entire Internet. In practice, as revealed by Snowden, this is what the NSA has already done [Edg17]. On the other hand, standard network protocols (i.e., TCP/IP) give us some assurance that message packets are not dropped on the links, so it is reasonable to model the adversary the way that we do, and, in fact, this is the standard model in the cryptography literature [Gol98].

Informally, a protocol is anonymous if the standard adversary cannot distinguish the scenario in which Anna sends to Roberto (scenario 0) from the scenario in which Anna sends to Leo instead (scenario 1) or, alternatively, the scenario in which Anna is not sending a message to anyone (scenario 2). For formal definitions, see Definitions 2 and 10.

In recent years, a few efficient (having low communication complexity) and provably secure onion routing protocols have been proposed [vdHLZZ15, TGL+17, KCDF17]. However,

[3] While the original attack by Øverlier and Syverson [ØS06] assumed a static set up (without Internet churn), recent attacks (see Raptor attacks [SEV+15] and Tempest attacks [WSJ+18]) exploit the dynamic nature of the Internet to increase both the adversarial view as well as the adversary's deanonymizing capabilities. Using such attacks, Sun et al. demonstrated that approximately 30% of Tor circuits can be deanonymized by an AS within three weeks [SEV+15].

[4] Compared with Tor, our protocols have longer latency. This is unavoidable since it was recently shown that an honestly formed onion must pass through a polylogarithmic (in the security parameter) number of intermediaries before being received by the recipient to provide anonymity from the passive adversary who monitors a constant fraction of the relays [Chr20].

[5] The names most commonly used for illustrating problems in cryptography and security are Alice and Bob who are users who wish to communicate securely, and Eve, the eavesdropper. In this thesis, we will use instead, the names of the thesis committee members, Anna, Roberto, Eli and Leo. Anna, Roberto and Eli naturally correspond to Alice, Bob and Eve since Anna begins with an "A", Bob is short for Robert and Eli begins with an "E".

these protocols are still impractical either because they send all messages through the same set of dedicated servers [vdHLZZ15] or are *fragile* (no one receives his/her message if any one message packet is dropped) [TGL+17, KCDF17]. In Chapters 2 and 3, we will present our anonymous onion routing protocols that are efficient, fault-tolerant and scalable (i.e., properly load-balanced).

Our contributions Suppose that Eli suspects that Anna is streaming communication to Roberto. If Eli is an active adversary, he may try to disrupt Anna's communication by dropping all the onions Anna emits. If Roberto then stops receiving (as many) onions, Eli can infer that his suspicion was correct: Anna is communicating with Roberto! In the asynchronous communications model, disrupting communication is "easy": Eli can guarantee that no onion formed by Anna makes it to the recipient. Likewise, even in the synchronous communication model, if Eli can adaptively choose which nodes to corrupt, disrupting communication is still "easy". For every onion o sent by Anna, Eli can corrupt the party p who receives o in time to direct p to drop the onion obtained from processing o before the next round.

As anonymity in the asynchronous or adaptive setting is hopeless, in this thesis, we examine the required number of onion transmissions for achieving anonymity in the synchronous and non-adaptive setting. Specifically, our protocols are secure whenever corruptions at round r do not depend on the view at round r. This is reasonable given the brevity of a single round. For our lower bound, assuming non-adaptivity does not lead to any loss of generality.

Like all previous results [vdHLZZ15,TGL+17,KCDF17] in provably secure onion routing, our results are for the *simple input-output (simple I/O) setting* in which each participant desires to send a fixed-length message to a unique interlocutor. Our contributions are as follows:

1. An anonymous protocol, Π_p, with polylogarithmic (in the security parameter) communication complexity (multiplicative) blowup in the presence of the passive adversary who can monitor a constant fraction of the nodes. This result can also be thought of as provable mixing (hiding the senders/origins) in the passive adversary setting. See §2.5.

2. A differentially private protocol, Π_a, with polylogarithmic (in the security parameter) communication complexity blowup in the presence of the active adversary who can control a constant fraction of the nodes. This result can also be thought of as provable

mixing in the active adversary setting. See §2.6.

3. The first onion routing protocol, $\Pi_{\bowtie}$, that is simultaneously efficient, fault tolerant, and anonymous in the active adversary setting. The protocol requires an expected polylogarithmic number of onions to be transmitted per message. See §3.5.

4. A lower bound (matching 3) for achieving anonymity in the active adversary setting. We show that for an onion routing protocol to be anonymous and fault-tolerant, there exists a sender who sends a polylogarithmic number of onions. See §3.4.

These results were published as separate papers [ALU18, ALU19].

1.2 Onion encryption (OE) schemes

While Chapters 2 and 3 are about onion routing protocols (e.g., hiding the communication patterns from the traffic flow in networks), Chapter 4 is about onion encryption schemes. Informally, an onion encryption scheme is a suite of algorithms for forming and processing onions, $(G, \mathsf{FormOnion}, \mathsf{ProcOnion})$, with useful properties for ensuring anonymity when used in an onion routing protocol. The algorithm G generates key pairs for the participants. The algorithm $\mathsf{FormOnion}$ forms an onion from the message and routing information, and the algorithm $\mathsf{ProcOnion}$ peels an onion and returns the peeled onion.

Before we describe what makes an onion encryption scheme correct and secure (in Chapter 4), we first explain why a public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *not* an ideal onion encryption scheme.

Suppose that Anna wants to form an onion for Roberto that routes through Carol and David before reaching Roberto. First, Anna encrypts her message m under Roberto's public key pk_B. The resulting ciphertext $c \leftarrow \mathsf{Enc}(\mathsf{pk}_B, m)$ is what she wishes Roberto to receive. Next, Anna encrypts the instructions "send c to Roberto" under David's public key pk_D, resulting in the ciphertext $c_D \leftarrow \mathsf{Enc}(\mathsf{pk}_D, \text{"send } c \text{ to Roberto"})$. Next, she encrypts the instructions "send c_D to David" under Carol's public key pk_C, resulting in the ciphertext $c_C \leftarrow \mathsf{Enc}(\mathsf{pk}_C, \text{"send } c_D \text{ to David"})$. In our example, the onion is c_C, and each node can decrypt the appropriate layer until Roberto receives the intended ciphertext c. Each mix-server, Carol or David, knows only the identities of adjacent nodes on the routing path. The above is essentially how Chaum described onion routing in the seminal 1981 paper introducing mixes [Cha81].

However, as noted by Camenisch and Lysyanskaya (CL) [CL05], using a public key encryption scheme in this way can reveal how far an onion is from the recipient (in hops).

This is because the size of the onion necessarily grows with the number of layers it has. Revealing this information is not ideal since, in this case, only onions with the same number of remaining layers will mix at honest nodes. In their seminal work on onion routing [CL05], Camenisch and Lysyanskaya address this issue; they formally defined onion encryption schemes, provided the first description of an ideal functionality $\mathcal{F}_{\mathsf{onion}}$ of onion encryption schemes in the universal composability (UC) model [Can01] and presented a construction that UC-realizes $\mathcal{F}_{\mathsf{onion}}$.

While CL provided a formal treatment of onion encryption schemes for one-way anonymous channels, they left open the problem of onion encryption for two-way anonymous channels. An onion formed from an onion encryption scheme for one-way communication is not "repliable" in that the onion's recipient is unable to respond to the onion's anonymous sender. In contrast to this, an onion encryption scheme for two-way communication includes an additional algorithm, $\mathsf{FormReply}$, for forming a return onion O' from a reply message m and the onion O received by the recipient. For this reason, we shall call such an onion encryption scheme, a *repliable onion encryption scheme*.

Despite the widespread use of two-way communication channels, for example, web browsing, no formalism for repliable onion encryption schemes existed. The popular Tor network allows the recipient to respond to the anonymous sender through a Tor circuit (a two-way channel established over TLS) [DMS04]. However, Tor is not anonymous in the standard adversary model, in large part because creating and maintaining Tor circuits is "leaky". Chaum presented a reply option (a mechanism for the recipient to reply to the sender) in his 1981 paper [Cha81]. However, as mentioned previously, this solution is not cryptographically secure either. Babel [GT96], Mixminion [DDM03], Minx [DL04] and Sphinx [DG09] also provide a reply option but also don't provide any formal definition or rigorous proof of security. This left a gap between proposed ideas for a repliable onion encryption scheme and rigorous examinations of these ideas. For instance, Kuhn et al. [KBS19] pointed out a fatal security flaw in the current art-of-the-art, Sphinx. They also pointed out some definitional issues in CL's and proposed fixes for some of these issues but left open the problem of formalizing repliable onion encryption.

Our contributions In this thesis, we formally define and realize repliable onion encryption. Our results are as follows:

1. The first description of an ideal functionality, $\mathcal{F}_{\mathsf{ROES}}$, for repliable onion encryption schemes. We describe $\mathcal{F}_{\mathsf{ROES}}$ in the simplified universal composability (SUC)

model [CCL15] as opposed to the full-blown UC model as this choice simplifies how the parties communicate with one another. The description of the ideal functionality $\mathcal{F}_{\mathsf{ROES}}$ can be found in §4.3.

2. The first provably secure repliable onion encryption scheme: Shallot Encryption Scheme. Our scheme builds on a CCA2-secure cryptosystem with tags, a block cipher and a collision-resistant hash function. We give our construction in §4.4 and prove that it SUC-realizes $\mathcal{F}_{\mathsf{ROES}}$ in §4.5.

These results were published as a separate paper [AL20].

Chapter 2

Simple and Provably Secure OR Protocols

2.1 Overview and related work

Tor The Tor network [DMS04] is a distributed network consisting of thousands of relay nodes and used by two million users per day[1] to communicate anonymously. Tor is based on a highly efficient design that favors practicality over provable security. It has low latency firstly, because onions are sent to their next destinations without delay, and, secondly, because each Tor circuit (i.e., routing path) consists of only three relay nodes (i.e., intermediaries) by default. The Tor architecture sacrifices provable security for low latency since it does not ensure that onions are "mixed" together and so cannot guarantee anonymity from the network adversary with a full view of the network traffic. Moreover, if an honestly formed onion is routed through only a constant number of intermediaries, then the probability that all the intermediaries are corrupted parties is non-negligible (in the security parameter).

2.1.1 Known provably secure OR protocols

On the flip side, researchers have known about provably secure methods that are too impractical in real systems. For example, Rackoff and Simon [Rac93] and Raymond [Ray01] pointed out that perfect secrecy can be achieved by broadcasting every message, thereby increasing the communication complexity by a linear (in the size of the network) factor. However, this approach is computationally expensive and only protects from the network

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

adversary who observes the traffic on links but doesn't corrupt nodes but doesn't protect from the passive or active adversary.

Much more recently, researchers have proposed onion routing protocols that are both efficient and provably secure; these are Vuvuzela [vdHLZZ15], Stadium [TGL+17] and Atom [KCDF17]. These protocols are secure in the setting where communication is modeled to be synchronous. In Vuvuzela and in Stadium, parties are either communicating in pairs or talking to themselves. In Atom, every party is sending a short message to a public bulletin board.

Vuvuzela Vuvuzela is a communications protocol suite by van den Hooff et al., consisting of a conversation protocol (for communicating messages) and a dialing protocol (for setting up who is communicating with whom) [vdHLZZ15]. Each user is assumed to behave honestly and is communicating with one party; Anna is sending a message to Roberto if and only if Roberto is sending a message to Anna. During setup, each user forms an onion using a message, the dedicated sequence $S = (S_1, S_2, \ldots, S_N)$ of mix-servers and the public keys associated with the servers.

At every round $i \in [N-1]$ of the protocol run, server S_i receives an ordered sequence of onions from the users or the previous server S_{i-1}. S_i peels the onions, injects some dummy onions to the sequence, randomly permutes the sequence of onions and, at the next round, sends the permuted sequence of onions to the next server S_{i+1}. When the last server S_N receives a sequence of onions, at round N, S_N peels the onions revealing a "dead-drop" address for each onion. By design, two onions share a dead-drop address, with overwhelming probability, only if a communicating pair formed these onions. S_N switches the order of every pair of onions with the same dead-drop address and, at round $N+1$, sends the sequence of onions back to the previous server S_{N-1}. At each round $i \in \{N+1, \ldots, 2N-1\}$ (in the reverse direction from S_N back to the users), S_{2N-i} un-permutes the sequence of onions, drops the dummy onions that it added in the forward direction, peels the remaining onions and sends the new sequence of onions to the next server S_{2N-i-1}. If the server is the first server S_1, it sends the peeled onions to their intended recipients (i.e., the users).

Vuvuzela's conversation protocol is secure when at least one of the servers is honest. The presence of the honest server guarantees that the adversary cannot map the onions sent by the users to onions received by the users. Through the injection of dummy onions, the number of communicating pairs is made differentially private from the (possibly corrupt) last server S_N. However, each server must receive, process and transmit all messages (along with dummy onions). This creates congestion in the network.

Stadium Stadium is an OR protocol by Tyagi et al. [TGL+17] that "load-balances" Vuvuzela. Rather than passing through a dedicated sequence of mix-servers, in Stadium, the onions pass through different sequences of groups of servers. Each group of servers is large enough to contain at least one honest party (with overwhelming probability), and the group's job is to shuffle incoming onions and split the outgoing onions to be received by different groups of servers in the next round. Stadium relies on verifiable shuffling [Abe98] to ensure that corrupted nodes cannot cheat by dropping onions or biasing the locations of outgoing onions.

Atom Atom is an anonymous messaging protocol for microblogging [KCDF17]. Like in Stadium, each "node" of Atom's overlay network is a group of servers that is guaranteed to include at least one honest server (with overwhelming probability); such a group of servers is referred to as an *anytrust group*. Atom's overlay network is a random permutation network, e.g., a butterfly network or a square network. So long as the anytrust groups behave honestly, this ensures that the messages sent through the network provably mix. When an anytrust group receives a set of onions, the servers in the group collectively peel the incoming onions and shuffle and batch the peeled onions. Like in Stadium, the protocol for doing this relies heavily on verifiable shuffling, with the idea that an honest server in the group will detect any cheating (modifying or dropping onions) by a corrupted party.

In contrast to Vuvuzela, Stadium and Atom are both properly load-balanced. However, both are highly *fragile*; if a corrupted party drops a single onion, the protocol is aborted. In such a case, no user receives his/her message. A variant of Atom is more fault-tolerant but only achieves the much weaker k-anonymity rather than full anonymity.

2.1.2 Our contributions

In this chapter, we present two practical and provably secure onion routing protocols, Π_p and Π_a:

1. We prove that Π_p is anonymous (Definition 2) from the passive adversary capable of monitoring any constant fraction $\kappa \in [0, 1)$ of the servers when the server load (i.e., the number of outgoing onions per server per round) is $\Omega\left(\log^{1+\epsilon} \lambda\right)$ and the round complexity is $\Omega\left(\log^{1+\epsilon} \lambda\right)$, where $\epsilon > 0$ is any positive constant and λ is the security parameter. See Theorem 2.

2. We prove that Π_a is differentially private (Definition 5) from the active adversary capable of corrupting and controlling any constant fraction $\kappa \in [0, 1)$ of the parties

when the server load (i.e., the number of outgoing onions per party per round) is $\Omega\left(\log^{2(1+\epsilon)} \lambda\right)$ and the round complexity is $\Omega\left(\log^{1+\epsilon} \lambda\right)$, where $\epsilon > 0$ is any positive constant and λ is the security parameter. See Theorem 3.

To prepare onions, we use a cryptographic scheme that is strong enough that, effectively, the only thing that the active adversary can do with onions generated by honest parties is to drop them; (see the onion cryptosystem by Camenisch and Lysyanskaya [CL05] for an example of a sufficiently strong cryptosystem). Unfortunately, even with such a scheme, it is still tricky to protect Anna's privacy against an adversary that targets Anna specifically. Suppose that the adversary guesses that Anna is communicating with Roberto. If the adversary succeeds in blocking all of Anna's onions and not too many of the onions from other parties, and then Roberto may receive fewer onions than other parties, then the adversary's hunch that it was Anna will be confirmed.

How do we prevent this attack? For this attack to work, the adversary would have to drop a large number of onions — there is enough cover traffic in our protocol that dropping just a few onions does not do much. But once a large enough number of onions is dropped, the honest mix-servers will detect that an attack is taking place and will shut down before any onions are delivered to their destinations. Specifically, if enough onions survive half of the rounds, then privacy is guaranteed through having sufficient cover; otherwise, privacy is guaranteed because no message reaches its final destination with overwhelming probability. So the adversary does not learn anything about the destination of Anna's onions.

In order to make it possible for the mix-servers to detect that an attack is taking place, our honest users create "checkpoint" onions. These onions don't carry any messages; instead, they are designed to be "verified" by a particular mix-server in a particular round. These checkpoint onions are expected by the mix-server, so if one of them does not arrive, the mix-server in question realizes that something is wrong. If enough checkpoint onions are missing, the mix-server determines that an attack is underway and shuts down. Two different users, Anna and Allison, use a PRF with a shared key (this shared key need not be pre-computed, but can instead be derived from a discrete-log based public-key infrastructure under the decisional Diffie-Hellman assumption) in order to determine whether Anna should create a checkpoint onion that will mirror Allison's checkpoint onion. See Section 2.6.

Other related work Prior anonymity protocols with provable security tend to rely on heavy cryptographic machinery to achieve high security standards. Rackoff and Simon's

1993 paper offers the closest solution to ours. Like us, Rackoff and Simon show their solution is secure by proving a rapid convergence of Markov processes. Unlike our approach, an inflexible network architecture is used to make their security argument tractable. Additionally, Rackoff and Simon use secure multiparty computation (SMPC) for providing security against active adversaries. Thus, while Rackoff and Simon's solution for passive adversaries [Rac93] was shown to have a short $\mathcal{O}\left(\log^2 n\right)$ delay in a later paper by Czumaj [Czu99], their main protocol [Rac93] relies heavily on SMPC and is believed not to scale [Ray01]. Another earlier example of an anonymity protocol that uses SMPC but with a much narrower purpose, is the dining cryptographer's protocol, which Chaum first introduced as a means to study secure multiparty Boolean-OR computation [Cha88b]. Other related cryptographic tools used in constructing anonymity protocols include oblivious RAM (ORAM) and private information retrieval (PIR) [Coo95, Cor15]. Corrigan-Gibbs et al.'s Riposte solution makes use of a global bulletin board and has a latency of a couple of days [Cor15].

Comparatively, the only cryptographic primitives used in our constructions are a public-key encryption scheme and a pseudorandom function (PRF). Compared with earlier solutions designed to be secure against active adversaries, our protocol, Π_a, is arguably a more practical approach with added scalability from proper load-balancing and flexibility from dynamic routing.

Information-theoretically secure approaches provide alternatives to provably secure methods. Some notable examples in the literature include Berman et al.'s work on obscurant networks [Ber04], Köpf and Basin's analysis using guessing entropy [BFTS04], Chatzikokolakis et al.'s work on quantifying anonymity leakage in terms of channel capacities [CPP08], and work with min-entropy leakage by Dodis et al. [Dod04] and Alvim et al. [AAC+11].

2.2 Preliminaries

This section and the next (§2.3 Modeling the problem) pertain to Chapters 2 and 3.

2.2.1 Notation

For a set $\mathcal{S}$, we denote the cardinality of $\mathcal{S}$ by $|\mathcal{S}|$, and $s \leftarrow\!\!\$\, \mathcal{S}$ is an item from $\mathcal{S}$ chosen uniformly at random. For an algorithm $A(x)$, $y \leftarrow A(x)$ is the (possibly probabilistic) output y from running A on the input x. In this paper, $\log(x)$ is the logarithm of x base 2. For strings x and y, "$x||y$" is the concatenation of x and y.

We say that a function $f : \mathbb{N} \mapsto \mathbb{R}$ is negligible in the parameter λ, written $f(\lambda) = \mathsf{negl}(\lambda)$, if for a sufficiently large λ, $f(\lambda)$ decays faster than any inverse polynomial in λ. When λ is the security parameter, an event E_λ is said to occur with (non-)negligible probability if the probability of E_λ can(not) be bounded above by a function negligible in λ. An event occurs with overwhelming probability (abbreviated w.o.p.) if its complement occurs with negligible probability. We use the standard notion of a pseudorandom function [Gol01, Ch. 3.6].

2.2.2 OE schemes and OR protocols

Onion encryption schemes Our work on onion routing builds upon a secure onion encryption scheme. Recall that an *onion encryption scheme* [CL05] is a triple of algorithms: $(\mathsf{Gen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$. The algorithm Gen generates a public-key infrastructure for a set of parties. The algorithm $\mathsf{FormOnion}$ forms onions; and the algorithm $\mathsf{ProcOnion}$ processes onions.

Let $[N]$ be a set of participants. For every $i \in [N]$, let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ be the key pair generated for party $i \in [N]$, where λ is the security parameter.

Let $\mathcal{M}$ be the message space consisting of messages of the same fixed length, and let the nonce[2] space $\mathcal{S}$ consist of nonces of the same fixed length. $\mathsf{FormOnion}$ takes as input a message m from $\mathcal{M}$, an ordered list $(p_1, p_2, \ldots, p_{R+1})$ of parties from $[N]$, the public keys $(\mathsf{pk}_{p_1}, \mathsf{pk}_{p_2}, \ldots, \mathsf{pk}_{p_{R+1}})$ associated with these parties, and a list $(s_1, s_2 \ldots, s_R)$ of (possibly empty) strings that are nonces from $\mathcal{S}$ associated with layers of the onion. The party p_{R+1} is interpreted as the *recipient* of the message, and the list $(p_1, p_2, \ldots, p_{R+1})$ is the *routing path* of the message. The output of $\mathsf{FormOnion}$ is a sequence $(o_1, o_2, \ldots, o_{R+1})$ of onions. Such a sequence is referred to as an *evolution*, but every o_i in the sequence is an onion. Because it is convenient to think of an onion as a layered encryption object where processing an onion o_r produces the next onion o_{r+1}, we sometimes refer to the process of revealing the next onion as *decrypting the onion*, or *peeling the onion*. For every $r \in [R]$, only party p_r can peel onion o_r to reveal the next layer,

$$(p_{r+1}, o_{r+1}, s_{r+1}) \leftarrow \mathsf{ProcOnion}(\mathsf{sk}_{p_r}, o_r, p_r),$$

which contains the *peeled* onion o_{r+1}, the *next destination* p_{r+1}, and the nonce s_{r+1}. Only the recipient p_{R+1} can peel the innermost onion o_{R+1} to reveal the message,

$$m \leftarrow \mathsf{ProcOnion}(\mathsf{sk}_{p_{R+1}}, o_{R+1}, p_{R+1}).$$

[2]Here, a nonce is pseudorandom short number.

In our constructions, a sender of a message m to a recipient j "*forms an onion*" by generating nonces and running the $\mathsf{FormOnion}$ algorithm on the message m, a routing path $(p_1, p_2, \ldots, p_R, j)$, the public keys $(\mathsf{pk}_{p_1}, \mathsf{pk}_{p_2}, \ldots, \mathsf{pk}_{p_R}, \mathsf{pk}_j)$ associated with the parties on the routing path, and the generated nonces; the *formed onion* is the first onion o_1 from the list of outputted onions. The sender (i.e., the party who formed the onion) sends o_1 to the first party p_1 on the routing path, who processes o_1 and sends the peeled onion o_2 to the next destination p_2 and so on, until the last onion o_{R+1} is received by the recipient j who processes it to obtain the message m.

Let o^0 and o^1 be any two onions that only (honest) party i can process such that one of the onions was formed by an honest party using the message-recipient pair $(m^0, \mathsf{recipient}^0)$, and the other was formed by an honest party using the pair $(m^1, \mathsf{recipient}^1)$. Importantly, the adversary who can adaptively interact with the honest parties cannot tell which message-recipient pair produced which onion. See Camenisch and Lysyanskaya's paper [CL05] for formal definitions.

Onion routing protocols An *onion routing protocol* is a protocol in which all message packets passed between the participants of the protocol are treated as sets of cryptographic onions, i.e., the protocol directs parties to put on the outbound channels only outputs of $\mathsf{FormOnion}$ and $\mathsf{ProcOnion}$ and to apply $\mathsf{ProcOnion}$ to their incoming packets.

2.3 Modeling the problem

We consider a setting where N parties, labeled $1, 2, \ldots, N$, participate in an onion routing protocol. We assume that the protocol progresses in global rounds and that an onion sent at round r is received instantaneously at the same round r. We assume that the number N of participants and every other quantity in the protocol is polynomially bounded in the security parameter λ.

In this thesis, we analyze single runs of onion routing protocols.

Our results are for a standard setting [Gol98] where the adversary is active and non-adaptive: Unless stated otherwise, we consider the *active* adversary who can observe the traffic on *all* links and, additionally, can non-adaptively corrupt and control a *constant* fraction of the parties. By *non-adaptively* corrupt, we mean that the corruptions are made independently of any run. Once the adversary corrupts a party, she can observe the internal state and computations of the corrupted party and arbitrarily alter the behavior of the party. (Adversaries making *adaptive* corruptions are not considered since they can trivially block

network traffic from an honest party, thus making anonymity impossible.)

Let $\mathbb{A}_\kappa$ denote the set of all active adversaries who corrupt up to κ fraction of the parties.

Inputs An input $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ to the protocol is a vector of inputs to the parties, where σ_i is a set of message-recipient pairs for party i. For $m \in \mathcal{M}$ and $j \in [N]$, the inclusion of a message-recipient pair (m, j) in input σ_i means that party i is instructed to send message m to recipient j.

Let Σ be a set of input vectors. Let $\mathcal{A}$ be the adversary, and let Bad be the set of parties controlled by $\mathcal{A}$. Fixing Bad imposes an equivalence class on Σ: Each equivalence class is defined by a vector $(d_1, d_2, \ldots, d_N)$. For each corrupted party $i \in \mathsf{Bad}$, $d_i = (\sigma_i, O_i)$ "fixes" the input σ_i for i and also, the set O_i of messages instructed to be sent from honest parties to i. For each honest party $i \in [N] \setminus \mathsf{Bad}$, $d_i = V_i$ "fixes" the number V_i of messages instructed to be sent from honest parties to i. An input vector belongs to the equivalence class $(d_1, d_2, \ldots, d_N)$ if for every $i \in \mathsf{Bad}$, $d_i = (\sigma_i, O_i)$, the input for i is σ_i and the set of messages from honest parties to i is O_i; and if for every $i \in [N] \setminus \mathsf{Bad}$, $d_i = V_i$, and the number of messages from honest parties to i is V_i. Two input vectors σ^0 and σ^1 are equivalent w.r.t. the adversary's choice Bad for the corrupted parties, denoted $\sigma^0 \equiv_{\mathsf{Bad}} \sigma^1$, if they belong to the same equivalence class imposed by Bad.

Outputs and views Let $\$ = (\$_1, \$_2, \ldots, \$_N, \$_{\mathcal{A}})$ denote the vector of random tapes used by the N participants and the adversary $\mathcal{A}$.

By $\mathsf{O}^{\Pi(1^\lambda,\mathsf{pp},\mathsf{states},\$),\mathcal{A}}(\sigma) = \left(\mathsf{O}_i^{\Pi(1^\lambda,\mathsf{pp},\mathsf{states},\$),\mathcal{A}}(\sigma)\right)_{i\in[N]}$, we mean the outputs (for honest parties, the messages received) for the N parties from running protocol Π interacting with adversary $\mathcal{A}$ with public parameters "pp" and internal states "states" on input the security parameter λ and vector σ of inputs, using vector $\$$ for random tapes. In general, when talking about multiple statistical quantities in a run of protocol Π, we implicitly mean that the quantities are conditioned on the same setting (i.e., pp, states and $\$$). We will, therefore, often use the shorthand $\mathsf{O}^{\Pi,\mathcal{A}}(\sigma)$.

For a given adversary $\mathcal{A}$, we abbreviate by $\mathsf{V}^{\Pi,\mathcal{A}}(\sigma)$, the adversary's view from interacting with protocol Π on input σ. The view consists of all the onions on every wire of network, the states of the corrupted parties, the randomness used by the adversary and, additionally, *the numbers of messages received by the honest parties.* We will sometimes denote by $\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma)$, the adversary's view conditioned on her choice Bad for the corrupted parties (in other words, when the adversary's randomness for choosing the corruptions is fixed but not the randomness for the other random choices in the system).

We define correctness w.r.t. interactions with the *passive* adversary as follows.

Definition 1 (Correctness)**.** *A communications protocol* Π *is correct if for any input* σ *and for every recipient* $j \in [N]$*,* j*'s output* $\mathsf{O}_j^{\Pi,\mathcal{A}_p}(\sigma)$ *in a run of* Π *interacting with the passive adversary* $\mathcal{A}_p$ *corresponds to the multiset of all messages for* j *in* σ*. That is,*

$$\mathsf{O}_j^{\Pi,\mathcal{A}}(\sigma_p) = \{m \,|(m,j) \in M(\sigma)\},$$

where $M(\sigma)$ *denotes the multiset of all message-recipient pairs in* σ*.*

2.4 Definitions

2.4.1 Statistical definitions of anonymity

For the analyses in this chapter, we assume an idealized version of an encryption scheme, in which the ciphertexts are information-theoretically unrelated to the plaintexts that they encrypt and reveal nothing but the length of the plaintext. When used in forming onions, such an encryption scheme gives rise to onions that are information-theoretically independent of their contents, destinations and identities of the mix-servers. The standard and natural notion of security under this model is,

Definition 2 (Statistical anonymity)**.** *A protocol* $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$ *is statistically anonymous from the adversary* $\mathcal{A}$ *who corrupts up to a constant* $0 \leq \kappa < 1$ *fraction of the parties for input vectors from the set* Σ *if for any choice* Bad *for corrupted parties such that* $|\mathsf{Bad}| \leq \kappa N$ *and for any pair* $\sigma^0, \sigma^1 \in \Sigma$ *of inputs such that* $\sigma^0 \equiv_{\mathsf{Bad}} \sigma^1$*, the adversary's views* $\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^0)$ *and* $\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^1)$ *are statistically indistinguishable, i.e.,*

$$\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^0) \approx_s \mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^1).$$

Π *is. perfectly secure from* $\mathcal{A}$ *if* $\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^0) = \mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^1)$ *instead.*

Definition 3 (Distance between inputs)**.** *The distance* $d(\sigma^0, \sigma^1)$ *between two inputs* $\sigma^0 = (\sigma_1^0, \ldots, \sigma_N^0)$ *and* $\sigma^1 = (\sigma_1^1, \ldots, \sigma_N^1)$ *is given by*

$$d(\sigma^0, \sigma^1) \stackrel{\text{def}}{=} \sum_{i=1}^{N} |\sigma_i^0 \nabla \sigma_i^1|,$$

where $(\cdot\nabla\cdot)$ *denotes the symmetric difference.*

Definition 4 (Neighboring inputs)**.** *Two inputs* σ^0 *and* σ^1 *are neighboring inputs if* $d(\sigma^0, \sigma^1) \leq 1$*.*

We also consider a weaker notion of anonymity, which we call differential anonymity (abbreviated DA) since it is essentially Dwork et al.'s differential privacy definition [DR14, Definition 2.4].

Definition 5 ((ϵ, δ)-DA)**.** *A protocol* $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$ *is* (ϵ, δ)*-DA from the adversary* $\mathcal{A}$ *who corrupts up to a constant* $0 \leq \kappa < 1$ *fraction of the parties for input vectors from the set* Σ *if for any choice* Bad *for corrupted parties such that* $|\mathsf{Bad}| \leq \kappa N$*, for any pair* $\sigma^0, \sigma^1 \in \Sigma$ *of inputs such that* $\sigma^0 \equiv_{\mathsf{Bad}} \sigma^1$ *and for any set* $\mathcal{V}$ *of views,*

$$\Pr\left[\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^0) \in \mathcal{V}\right] \leq e^\epsilon \cdot \Pr\left[\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^1) \in \mathcal{V}\right] + \delta.$$

While differential privacy is defined with respect to neighboring inputs, it also provides, albeit weaker, guarantees for non-neighboring inputs; it is known that the security parameters degrade proportionally in the distance between the inputs [DR14].

2.4.2 Efficiency measures

Our main measure of efficiency of OR protocols is *onion cost*, Our measure of efficiency of OR protocols is *onion cost*, which measures how many onions are transmitted by each party in the protocol.

Let $\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma, r)$ denote the number of honestly formed onions that party i transmits directly (to another party) in the r-th round of a protocol run of Π interacting with adversary $\mathcal{A}$ on input vector σ, and let $\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma)$ denote the total number of honestly formed onions that party i transmits directly (to another party) in a protocol run of Π interacting with adversary $\mathcal{A}$ on input vector σ.

Definition 6 (Onion cost)**.** *For an OR protocol* $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$*, an adversary* $\mathcal{A}$ *and an input set* Σ*, the onion cost of* Π *interacting with* $\mathcal{A}$ *w.r.t.* Σ *is the expected number of honest onions transmitted per party in a run of* Π *interacting with* $\mathcal{A}$ *on a random input* $\sigma \leftarrow_\$ \Sigma$ *from* Σ*, i.e.,*

$$\mathsf{OC}^{\Pi,\mathcal{A}}(\Sigma) \stackrel{\text{def}}{=} \mathbb{E}_{\sigma,i,\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma)\right].$$

The expectation is taken over the uniformly random input $\sigma \leftarrow_\$ \Sigma$*, the uniformly random party* $i \leftarrow_\$ [N]$ *and the randomness* $\$$ *of the protocol.*

For an adversary class $\mathbb{A}$*, the onion cost of* Π *interacting with* $\mathbb{A}$ *w.r.t.* Σ *is the maximum onion cost over the adversaries in* $\mathbb{A}$*, i.e.,* $\mathsf{OC}^{\Pi,\mathbb{A}}(\Sigma) \stackrel{\text{def}}{=} \max_{\mathcal{A}\in\mathbb{A}} \mathsf{OC}^{\Pi,\mathcal{A}}(\Sigma)$.

The onion cost is measured in unit onions, which is appropriate when the parties pass primarily onions to each other.

It is also an attractive measure of complexity because it is algorithm-independent: If we measured complexity in bits, it would change depending on which underlying encryption scheme was used. Since an onion contains as many layers as there are intermediaries, its bit complexity scales linearly with the number of intermediaries. (We assume that every message m can be contained in a single onion.) To translate our lower bound from onion complexity to bits, we will consider onions to be at least as long (in bits) as the message m being transmitted and the routing information.

Definition 7 (Server load)**.** *For an OR protocol* $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$*, an adversary* $\mathcal{A}$ *and an input set* Σ*, the server load of* Π *interacting with* $\mathcal{A}$ *w.r.t.* Σ *is the expected number of honest onions transmitted per party per round in a run of* Π *interacting with* $\mathcal{A}$ *on a random input* $\sigma \leftarrow_\$ \Sigma$ *from* Σ*, i.e.,*

$$\mathsf{SL}^{\Pi,\mathcal{A}}(\Sigma) \stackrel{\text{def}}{=} \mathbb{E}_{\sigma,i,r,\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma, r)\right].$$

The expectation is taken over the uniformly random input $\sigma \leftarrow_\$ \Sigma$*, the uniformly random party* $i \leftarrow_\$ [N]$*, the uniformly random round and the randomness* $\$$ *of the protocol.*

Definition 8 (Round complexity)**.** *For an OR protocol* $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$*, an adversary* $\mathcal{A}$ *and an input set* Σ*, the server load of* Π *interacting with.* $\mathcal{A}$ *w.r.t.* Σ *is the expected number of rounds in a run of* Π *interacting with* $\mathcal{A}$ *on a random input* $\sigma \leftarrow_\$ \Sigma$ *from* Σ*.*

In addition to having low (i.e., polylog in the security parameter) onion cost, we will show that our OR protocols have low (i.e., polylog in the security parameter) server load and low (i.e., polylog in the security parameter) round complexity.

2.5 Π_p, anonymity in the passive adversary setting

Like the active adversary, the passive adversary can observe all network traffic on all links and can corrupt a constant fraction of the nodes. Unlike the active adversary, the passive adversary does not control the corrupted nodes and merely monitors the internal states and computations of the corrupted nodes. Here, we present a simple OR protocol that provides anonymity from the passive adversary while being practical with low onion cost and low server load.

We assume that every user sends and receives the same number of messages as any other user; otherwise, the sender-receiver relation can leak from the differing numbers of messages sent and received by the users. In other words, every user essentially commits to sending a

message, be it the empty message $\perp$ to itself. Let $\mathsf{SimpleIO}$ be the set of all input vectors of the form

$$\sigma = (\{(m_1, \pi(1))\}, \ldots, \{(m_N, \pi(N))\}),$$

where $\pi : [N] \to [N]$ is a permutation function over the set $[N]$, and $m_1, \ldots, m_N$ are messages from the message space $\mathcal{M}$. We present our protocol Π_p in the setting in which the input vector is constrained to belong to $\mathsf{SimpleIO}$. We call this setting *the simple I/O setting*.

2.5.1 Description of Π_p

Let $[N]$ be the set of users and $\mathcal{S} = \{S_1, S_2, \ldots, S_n\} \subset [N]$, the set of servers. Π_p uses a secure onion encryption scheme, denoted $\mathcal{OE} = (\mathsf{Gen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$, as a primitive building block. For every $i \in [N]$, let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ be the key pair generated for party i, where λ denotes the security parameter. (See §2.2.2 for a description of onion encryption schemes.)

During setup, each user $i \in [N]$ creates an onion. On input $\sigma_i = \{(m, j)\}$, user i picks R servers $p_1, p_2, \ldots, p_R$ independently and and uniformly at random and then forms an onion using the message m, the routing path $(p_1, p_2, \ldots, p_R, j)$, the public keys $(\mathsf{pk}_{p_1}, \mathsf{pk}_{p_2}, \ldots, \mathsf{pk}_{p_R}, \mathsf{pk}_j)$ associated with the parties on the routing path and a list $(\perp, \perp, \ldots, \perp)$ of empty nonces.

1 : $p_1, \ldots, p_R \leftarrow [N]$
2 : $(o_1, \ldots, o_{R+1}) \leftarrow \mathsf{FormOnion}(m, (p_1, \ldots, p_R, j), (\mathsf{pk}_{p_1}, \ldots, \mathsf{pk}_{p_R}, \mathsf{pk}_j), (\perp, \ldots, \perp))$
3 : return o_1

Figure 2.1: In Π_p, instructions for forming an onion on input the security parameter 1^λ, the participants $[N]$, the set $\mathcal{S} \subseteq [N]$ of servers, the participants' public keys, the message m and the recipient j.

At the first round of the protocol run, user i sends the formed onion to the first server p_1 on the routing path. After every round $r \in [R]$ (but before round $r+1$) of the protocol run, each server processes the onions it received at round r. At round $r+1$, the resulting peeled onions are sent to their respective next destinations in random order. At round $R+1$, every user receives an onion and processes it to reveal a message.

Correctness and efficiency Clearly, Π_p is correct. In Π_p, N messages are transmitted in each of the $R+1$ rounds of the protocol run. Thus, the round complexity is $R+1$, and the onion cost is $\mathcal{O}(N(R+1))$. The server load is $\mathcal{O}\left(\frac{N}{n}\right)$.

2.5.2 Proof that Π_p is anonymous

To prove that Π_p is anonymous (Definition 2) from the passive adversary, we first prove that it mixes onions in the presence of the network adversary, i.e., the adversary who can observe all network traffic but cannot corrupt any nodes.

Theorem 1. *Π_p is anonymous (Definition 2) from the network adversary when the server load is $\Omega\left(\log^{1+\epsilon}\lambda\right)$ and the round complexity is $\Omega\left(\log^{1+\epsilon}\lambda\right)$, i.e., $\frac{N}{n} = \Omega\left(\log^{1+\epsilon}\lambda\right)$ and $R = \Omega\left(\log^{1+\epsilon}\lambda\right)$.*

Proof. Let $\frac{N}{n} = \alpha \log^{1+\epsilon}\lambda$ for $\alpha = \Omega(1)$; so that after each round, every location receives $\alpha \log^{1+\epsilon}\lambda$ onions in expectation. We recast our problem as a balls-in-bins problem, where the balls are the onions and the bins, the locations. At every round of the protocol, all $\alpha n \log^{1+\epsilon}\lambda$ balls (i.e., onions) are thrown uniformly at random into n bins (i.e., each onion is routed to one of n locations chosen independently and uniformly at random).

Fix any target sender U, and let $X^r = (X_1^r, \ldots, X_n^r)$ be a vector of non-negative numbers summing to one, representing $\mathcal{A}$'s best estimate for the location of U's ball after r rounds (and before round $r+1$); for every $i \in [N]$, X_i^r is the likelihood that bin i contains U's ball after r rounds. Let $(X^r_{f_r(1)}, \ldots, X^r_{f_r(n)})$ be the result of sorting $(X_1^r, \ldots, X_n^r)$ in decreasing order, where $f_r : [N] \to [N]$ is a permutation function over the set $[N]$. For every $i \in [N]$, let $b_i^r = f_r(i)$ be the index of the bin with the i-th largest likelihood at round r.

W.l.o.g. we assume that n is divisible by three. We partition the bins into three groups G_1^r, G_2^r and G_3^r; such that G_1^r contains all the *balls* in the top one-third most likely bins $b_1^r, \ldots, b^r_{\frac{n}{3}}$; G_3^r contains all the balls in the bottom one-third most likely bins $b^r_{\frac{2n}{3}+1}, \ldots, b_n^r$; and G_2^r contains all the balls in the remaining bins $b^r_{\frac{n}{3}+1}, \ldots, b^r_{\frac{2n}{3}}$.

For each $j \in [3]$, let $O_j^r \in G_j^r$ be a ball with the maximum likelihood of being U's onion among the balls in group G_j^r. For any $d \in (0,1)$, let $d' = 1 - \frac{1}{1+d}$. Let c_j^r be the bin containing O_j^r. The bin c_j^r contains at least $(1-d')\alpha \log^{1+\epsilon}\lambda$ balls (Chernoff bounds for Poisson trials [MU05]). It follows that with overwhelming probability,

$$\Pr\left[O_j^r \text{ is } U\text{'s onion}\right] \leq (1+d)\frac{X^r_{c_j^r}}{\alpha \log^{1+\epsilon}\lambda} \leq (1+d)\frac{X^r_{\frac{(j-1)n}{3}+1}}{\alpha \log^{1+\epsilon}\lambda}, \tag{2.1}$$

where $X^r_{\frac{(j-1)n}{3}+1}$ is the likelihood of the most likely bin in group G_j.

The number of balls contained in each group G_j^r is arbitrarily close to the expected number $\frac{\alpha}{3} n \log^{1+\epsilon}\lambda$ of balls in a group (Chernoff bounds). Thus, the most probable bin b_1^{r+1} after the next round receives at most $\frac{(1+d)\alpha}{3}\log^{1+\epsilon}\lambda$ balls from each of the three groups:

G_1^r, G_2^r and G_3^r. From (2.1), this implies that with overwhelming probability,

$$X_1^{r+1} \leq \frac{(1+d)^2}{3} \sum_{j=1}^{3} X^r_{\frac{(j-1)n}{3}+1}.$$

Using a symmetric argument, we can conclude that with overwhelming probability,

$$X_n^{r+1} \geq \frac{(1-d)^2}{3} \sum_{j=1}^{3} X^r_{\frac{jn}{3}},$$

where $X^r_{\frac{jn}{3}}$ is the likelihood of the least likely bin in group G_j.

For all $r \in [R]$, define $g^r = X_1^r - X_n^r$ as the difference in likelihoods between the most and least likely bins at round r.

$$g^{r+1} \leq \frac{(1+d)^2 \sum_{j=1}^{3} X^r_{\frac{(j-1)n}{3}+1}}{3} - \frac{(1-d)^2 \sum_{j=1}^{3} X^r_{\frac{jn}{3}}}{3} \leq \frac{1}{2}\left(X_1^r - X_n^r\right) = \frac{g^r}{2},$$

where the latter inequality follows from telescopic cancelling, since

$$X^r_{\frac{n}{3}+1} \leq X^r_{\frac{n}{3}},$$

and

$$X^r_{\frac{2n}{3}+1} \leq X^r_{\frac{2n}{3}}.$$

The difference g^r is at least halved at every round. By round $\log^{1+\epsilon} \lambda$, the difference is negligible in λ. Thus, after traveling R random hops, each onion becomes "unlinked" from its sender.

In the proof above, the bins were partitioned into three groups at every round. By partitioning the bins into an appropriately large constant number of groups, we can show that Π_p achieves statistical privacy after $R = \Omega\left(\log^{1+\epsilon} \lambda\right)$ rounds. □

We are now ready to prove the main result of this section:

Theorem 2. *Π_p is anonymous (Definition 2) from the passive adversary capable of monitoring any constant fraction $\kappa \in [0, 1)$ of the servers when $\frac{N}{n} = \Omega\left(\log^{1+\epsilon} \lambda\right)$ and $R = \Omega\left(\log^{1+\epsilon} \lambda\right)$, where $\lambda \in \mathbb{N}$ denotes the security parameter.*

Proof. We prove this by cases.

In the first case, σ^1 is the same as σ^0 except that the inputs of two users are swapped, i.e., $d(\sigma^0, \sigma^1) = 2$. Using Chernoff bounds for Poisson trials, there are at least some polylog

number of rounds where the swapped onions are both routed to an honest bin (not necessarily the same bin). From Theorem 1, after the polylog number of steps, the locations of these two target onions are statistically indistinguishable from each other.

In the second case, $d(\sigma^0, \sigma^1) > 2$. However, the distance between σ^0 and σ^1 is always polynomially bounded. By a simple hybrid argument, it follows that $\mathsf{V}^{\Pi,\mathcal{A}}(\sigma^0) \approx_s \mathsf{V}^{\Pi,\mathcal{A}}(\sigma^1)$ from case 1. □

2.6 Π_a, differential anonymity in the active adversary setting

We present an OR protocol, Π_a, that is secure from the active adversary. The setting for Π_a is different from that of Π_p (Section 2.5) in a few important ways. Whereas Π_p is fully anonymous from the passive adversary, Π_a is *differentially* anonymous from the *active* adversary. Like in Vuvuzela and Stadium, participants are talking in pairs or to themselves. The upside is each party can now send an arbitrary number of messages.

We let $[N]$ be the set of N parties participating in a protocol. Every party is both a user and a server. As before, $\mathcal{OE} = (\mathsf{Gen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$ is a secure onion encryption scheme; and for every $i \in [N]$, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ denotes the key pair generated for party i, where λ is the security parameter. Further, we assume that every pair $i, k \in [N]$ of parties share a common secret key,[3] denoted by $\mathsf{sk}_{i,k}$.

F is a pseudorandom function (PRF) [Gol01, Ch. 3.6].

2.6.1 Description of Π_a

We describe the protocol by the setup and routing algorithms for party $i \in [N]$; each honest party runs the same algorithms.

Setup Let $\alpha, \beta > 0$ be any positive constants such that the number of intermediaries per onion is $R = \beta \log^{1+\epsilon} \lambda$, and the expected server load is $\frac{N}{n} = \alpha \log^{1+\epsilon} \lambda + 1$. (These are system parameters that adjust the rate at which the errors vanish.)

During the setup phase, party i prepares a set of onions from its input.

For every message pair $\{m, j\}$ in party i's input, party i picks a sequence $p_1, p_2, \ldots, p_R$ of parties, where each party p_ℓ is chosen independently and uniformly at random, and forms an onion from the message m, the routing path $(p_1, p_2, \ldots, p_R, j)$, the public

[3] In practice, the shared keys do not need to be set up in advance; they can be generated as needed from an existing PKI, e.g., using Diffie-Hellman.

keys $(\mathsf{pk}_{p_1}, \mathsf{pk}_{p_2}, \ldots, \mathsf{pk}_{p_R}, \mathsf{pk}_j)$, and a list $(\perp, \perp, \ldots, \perp)$ of empty nonces. See lines 2-5 in the pseudocode below.

Party i also forms some checkpoint onions, where a checkpoint onion is an onion formed using the empty message $\perp$. For every party $k \in [N]$ and round $r \in [R]$, with fixed frequency $\frac{\alpha}{N} \log^{1+\epsilon} \lambda$, party i forms a checkpoint onion designed to reveal the checkpoint (i.e., the nonce value) $F(\mathsf{sk}_{i,k}, \mathsf{sid} + r, 1)$ to party k at round r. The nonce value is computed using a pseudorandom function F keyed with the shared key $\mathsf{sk}_{i,k}$. See lines 6-12 in the pseudocode below.

```
1 :  O = ∅
2 :  for each J ≝ {m, j} ∈ σ_i
3 :     p_1, ..., p_R ← [N]
4 :     (o_1, ..., o_{R+1}) ← FormOnion(m, (p_1, ..., p_R, j), (pk_{p_1}, ..., pk_{p_R}, pk_j), (⊥, ..., ⊥))
5 :     add o_1 to O
6 :  for (k, r) ∈ [N] × [R]
7 :     if F(sk_{i,k}, sid + r, 0) ≡ 1 (which occurs with frequency α log^{1+ε} λ/N)
8 :        p_1, ..., p_{R+1} ← [N]
9 :        s = (⊥, ..., ⊥)
10 :       s[r] = (ckpt, F(sk_{i,k}, sid + r, 1))
11 :       (o_1, ..., o_{R+1}) ← FormOnion(m, (p_1, ..., p_{R+1}), (pk_{p_1}, ..., pk_{p_{R+1}}), s)
12 :       add o_1 to O
13 :  return O
```

Figure 2.2: In Π_a, instructions for forming onions on input the security parameter 1^λ, the session id sid, the parties $[N]$, the parties' public keys and the input σ_i.

Routing By construction, party i forms a checkpoint onion to be expected by honest party k at round r iff party k forms a symmetric checkpoint onion to be expected by party i at round r. Each party (i or k) independently computes $b = F(\mathsf{sk}_{i,k}, \mathsf{sid} + r, 0)$ and, if $b \equiv 1$, forms a checkpoint onion with checkpoint $s = (\mathsf{ckpt}, F(\mathsf{sk}_{i,k}, \mathsf{sid} + r, 1))$. If party i forms a checkpoint onion with nonce s embedded in the r-th layer, then party i expects to receive a checkpoint onion at the r-th round that, when processed, reveals the same nonce s. If many expected checkpoint nonces are missing, then party i can deduce that something is wrong and aborts the protocol run.

After every round $r \in [R]$ (but before round $r + 1$), party i peels the onions it received at round r and counts the number of missing checkpoint nonces. If the count exceeds a threshold value t, the party aborts the protocol run; otherwise, at round $r + 1$, the peeled

onions are sent to their next destinations in random order.

After the final round, party i outputs the set of messages revealed from processing its the onions it receives at round $R + 1$.

Correctness and efficiency Recalling that correctness is defined with respect to the passive adversary (see Definition 14), Π_a is clearly correct.

Unless an honest party aborts the protocol run, all messages that are not dropped by the adversary are delivered to their final destinations. In Π_a, the communication complexity is $\mathcal{O}\left(N \log^{3(1+\epsilon)} \lambda\right)$, since the round complexity is $R + 1 = \mathcal{O}\left(\log^{1+\epsilon} \lambda\right)$ rounds, and the expected server load is $\mathcal{O}\left(\log^{2(1+\epsilon)} \lambda\right)$.

2.6.2 Proof that Π_a is differentially anonymous

To prove that Π_a is secure, we require that the thresholding mechanism does its job.

Lemma 1. *In Π_a: If F is a random function, $t = c(1-d)(1-\kappa)^2\alpha \log^{1+\epsilon} \lambda$ for some $c, d \in (0, 1)$, and an honest party does not abort within the first r rounds of the protocol run, then with overwhelming probability, at least $(1-c)$ of the checkpoint onions created between honest parties survive at least $(r-1)$ rounds, even in the presence of an active adversary who corrupts a constant $\kappa \in [0, 1)$ fraction of the parties.*

The proof relies on a known concentration bound for the hypergeometric distribution [HS05].

Proof. Let $\mathcal{A}$ be any active adversary capable of corrupting a constant $\kappa \in [0, 1)$ fraction of the parties. Since $\mathcal{A}$ controls the corrupted parties, she can know the checkpoint round, location, and nonce of any onion created between a corrupted party and any other party. Thus, we assume that any onion created between a corrupted party and any other party can be replaced by $\mathcal{A}$ without the replacement being detected by any honest party. Suppose that $\mathcal{A}$ has help from a dark angel who marks every onion created between a corrupted party and any other party, so that $\mathcal{A}$ can replace all marked onions without detection. Even so, without eliminating some unmarked onions, some positive constant fraction of the checkpoint onions would survive (Chernoff bounds).

Let an onion created between two honest parties be called *unmarked*, and consider only unmarked onions. For any onion with a checkpoint in the future, the probability that the adversary $\mathcal{A}$ can drop the onion without any honest party detecting that the onion was

dropped is negligibly small; $\mathcal{A}$ cannot produce the correct checkpoint nonce with sufficiently high probability.

At any round r, $\mathcal{A}$ is unable to distinguish between any two unmarked onions. Let u denote the total number of unmarked onions. Again relying on Chernoff bounds, with overwhelming probability

$$u \geq (1-d_1)(1-\kappa)^2 \alpha RN \log^{1+\epsilon} \lambda$$

for any $0 < d_1 < d$.

Let v_r denote the cumulative number of unmarked onions that have been eliminated by $\mathcal{A}$ so far at round r. If an honest party i does not detect more than

$$\frac{(1-d_2)cu}{RN} \geq (1-d_1)(1-d_2)c(1-\kappa)^2 \alpha \log^{1+\epsilon} \lambda = (1-d)c(1-\kappa)^2 \alpha \log^{1+\epsilon} \lambda$$

missing onions, then with overwhelming probability, $v_{r-1} \leq cu$ [HS05]. It follows that at least $1-c$ of all checkpoint onions created between honest parties survive until round $r-1$, with overwhelming probability. □

We now prove that Π_a is differentially anonymous from the active adversary.

Theorem 3. *In Π_a: If F is a random function, $N \geq \frac{3}{1-\kappa}$, and $t = c(1-d)(1-\kappa)^2 \alpha \log^{1+\epsilon} \lambda$ for some constants $c, d \in (0,1)$, then for*

$$\alpha\beta \geq -\frac{36(1+\epsilon/2)^2 \ln(\delta/4)}{(1-c)(1-\kappa)^2 \epsilon^2},$$

Π_a is (ϵ, δ)-DA from the active adversary who corrupts a constant fraction $\kappa \in [0,1)$ of the parties.

Proof. The proof is by cases.

Case 1: All honest parties abort within the first half of the protocol run. With overwhelming probability, no onion created by an honest party will be delivered to its final destination (Chernoff bounds), and so the adversary doesn't learn anything.

Case 2: Some honest party doesn't abort within the first half of the protocol run. Let $\mathcal{A}$ be any adversary that non-adaptively corrupts a constant $\kappa \in [0,1)$ of the parties. Suppose that for every onion that survive the first half of the protocol run, a dark angel provides $\mathcal{A}$ with the second half of the onion's routing path. Further, suppose that no other onions are dropped in the second half of the protocol run. (If more onions are dropped, then Π_a is secure from the post-processing theorem for differential privacy [DR14, Proposition 2.1].)

For any two neighboring inputs σ_0 and σ_1, the only difference in the adversary's views, $\mathsf{V}^{\Pi_a,\mathcal{A}}(\sigma_0)$ and $\mathsf{V}^{\Pi_a,\mathcal{A}}(\sigma_1)$, is the routing of a single onion O. If there is an honest party who does not abort within the first half of the protocol run, then from Lemma 1, some constant fraction of the checkpoint onions created by the honest parties survive the first half of the protocol run with overwhelming probability. So, from Theorem 2, the onions are no longer linked to their senders by the end of the first half of the protocol run. Thus, the only information that $\mathcal{A}$ could find useful is the volume of onions sent out by the sender p_s of the extra onion O and the volume of onions received by the receiver p_r of O.

Let X denote the number of checkpoint onions created by p_s. For every $(k, r) \in [R] \times [N]$, an honest sender p_s creates a checkpoint onion with probability $\frac{\alpha \log^{1+\epsilon} \lambda}{N}$; so $X \sim \mathsf{Binomial}(H, p)$, where $H = LN$, and $p = \frac{\alpha \log^{1+\epsilon} \lambda}{N}$.

Let $Y \sim \mathsf{Binomial}(G, q)$ be another binomial random variable with parameters $G = \frac{R(1-\kappa)^2 N^2}{3}$ and $q = \frac{(1-c)\alpha \log^{1+\epsilon} \lambda}{N^2}$. For $N \geq \frac{3}{1-\kappa}$ and sufficiently small $d > 0$, $G \leq (1 - d)L\binom{(1-\kappa)N-1}{2}$; thus, with overwhelming probability, Y is less than the number of checkpoint onions created between honest non-p_s parties and received by p_r in the final round (Chernoff bounds).

Let $\mathcal{O} \stackrel{\text{def}}{=} \mathbb{N} \times \mathbb{N}$ be the sample space for the multivariate random variable (X, Y).

Let $\mathcal{O}_1$ be the event that $|X - \mathbb{E}[X]| \leq d'\mathbb{E}[X]$, and $|Y - \mathbb{E}[Y]| \leq d'\mathbb{E}[Y]$, where $d' = \frac{\epsilon/2}{1+\epsilon/2}$, $\mathbb{E}[X] = Hp$ is the expected value of X, and $\mathbb{E}[Y] = Gq$ is the expected value of Y; and let $\bar{\mathcal{O}_1}$ be the complement of $\mathcal{O}_1$.

For every $(x, y) \in \mathcal{O}_1$, we can show that

$$\max\left(\frac{\Pr[(X,Y) = (x,y)]}{\Pr[(X,Y) = (x+1,y+1)]}, \frac{\Pr[(X,Y) = (x+1,y+1)]}{\Pr[(X,Y) = (x,y)]}\right) \leq e^{\epsilon}. \tag{2.2}$$

We can also show that the probability of the tail event $\bar{\mathcal{O}_1}$ occurring is negligible in λ and at most δ when $\alpha\beta \geq -\frac{36(1+\epsilon/2)^2 \ln(\delta/4)}{(1-c)(1-\kappa)^2\epsilon^2}$. (See the full version of this paper.)

Any event $\mathcal{E}$ can be decomposed into two subsets $\mathcal{E}_1$ and $\mathcal{E}_2$, such that (1) $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, (2) $\mathcal{E}_1 \subseteq \mathcal{O}_1$, and (3) $\mathcal{E}_2 \subseteq \bar{\mathcal{O}_1}$. It follows that, for every event $\mathcal{E}$,

$$\Pr[(X,Y) \in \mathcal{E}] \leq e^{\epsilon} \cdot \Pr[(X+1,Y+1) \in \mathcal{E}] + \delta, \text{ and} \tag{2.3}$$

$$\Pr[(X+1,Y+1) \in \mathcal{E}] \leq e^{\epsilon} \cdot \Pr[(X,Y) \in \mathcal{E}] + \delta. \tag{2.4}$$

The views $\mathsf{V}^{\Pi_a,\mathcal{A}}(\sigma_0)$ and $\mathsf{V}^{\Pi_a,\mathcal{A}}(\sigma_1)$ are the same except that O exists in one of the views but not in the other. Thus, (2.3) and (2.4) suffice to show that for any set $\mathcal{V}$ of views and for any $b \in \{0, 1\}$, $\Pr\left[\mathsf{V}^{\Pi_a,\mathcal{A}}(\sigma_b) \in \mathcal{V}\right] \leq e^{\epsilon} \cdot \Pr\left[\mathsf{V}^{\Pi_a,\mathcal{A}}(\sigma_{\bar{b}}) \in \mathcal{V}\right] + \delta$, where $\bar{b} = b + 1 \mod 2$. □

2.7 Concluding remarks

We presented practical (i.e., efficient, fault-tolerant and scalable) OR protocols that provably secure under standard adversary models. Our main construction, Π_a, is secure against an active adversary in control of a constant fraction $\kappa \in [0, 1)$ of the parties and with a global view of all traffic on all links in the network. It relies on a new technique for detecting when too many of the protocol's onions have been dropped and possibly replaced.

Before this work, the only known (load-balanced) way of guaranteeing the mixing of message packets was by routing them through a random permutation network (such as a butterfly network) in which every node behaves honestly [TGL+17, KCDF17].

In this chapter, we proved that a simple onion routing protocol, Π_p, can be anonymous from the passive adversary when the average server load (per server per round) and the round complexity are both polylogarithmic in the security parameter. Importantly, Π_p provides provable guarantees without requiring a rigid overlay network or honest behavior from every node in the network. Moreover, recent new work has since proved it to be asymptotically optimal in onion cost [Chr20].

However, Π_p is not secure from the active adversary. With reasonable probability, the active adversary can target Anna (a sender) by dropping her onion upfront. In such a case, the adversary can learn the recipient of Anna's message by observing who doesn't receive an onion at the last round. In the next chapter, we will construct an OR protocol $\Pi_{\bowtie}$ that achieves anonymity from the active adversary by building on the results from this chapter.

Chapter 3

Tight Bounds for OR Protocols

3.1 Overview and related work

In recent years, there has been progress towards provably secure onion routing protocols. However, the protocols proposed so far in the literature guarantee only differential privacy or k-anonymity rather than anonymity [vdHLZZ15, TGL+17, KCDF17, ALU18] or are not fault-tolerant [vdHLZZ15, TGL+17, KCDF17].

3.1.1 Our contributions

We initiate a rigorous theoretical study of onion routing by providing new definitions and both lower and upper bounds.

1. We focus on a natural game-based definition of anonymity (Definition 10), where an adversary cannot distinguish the scenario in which Anna sends a message to Roberto while Carol sends one to David from one in which Anna's message goes to David while Carol's goes to Roberto. We relate it to two new concepts: *equalizing* (i.e., whether or not each participant received a message, and if so, how many messages, is independent of the input to the protocol, Definition 11) and *mixing* (i.e., it is impossible to tell where an onion a recipient got at the end of the protocol originated from, Definition 12). First, we show that, in order to provide anonymity, an onion routing protocol must equalize (Theorem 6). This insight allows us to show our lower bound. Second, we show that an onion routing protocol that both equalizes and mixes provides anonymity; this is a key insight that guides the design and analysis of our protocol. Both the lower bound and the protocol we provide are for the *simple input-output (simple I/O) setting* in which each participant desires to send a fixed-length

message to a unique interlocutor.

2. We give a lower bound (Theorem 7): if an onion routing protocol Π over a point-to-point communication network is both fault-tolerant and anonymous in the presence of an active adversary controlling a constant κ fraction of parties, then every party in Π must transmit more than a logarithmic (in the security parameter) number of onions.

3. We also give a (near) matching upper bound (Theorem 8): we provide an onion routing protocol that provides anonymity against any fraction $\kappa < \frac{1}{2}$ of actively corrupted parties, in which honest participants transmit $\gamma_1 \log N \log^{3+\gamma_2} \lambda$ onions, where N is the number of participants, λ is the security parameter, and increasing γ_1 and γ_2 increases the rate at which the maximum distance in the adversarial views for any two inputs shrinks. This is the first protocol that is simultaneously anonymous and practical (fault-tolerant, distributed and with low communication complexity). For the construction, we introduce a new technique of *merging onions*.

Merging onions The chief stumbling block in achieving anonymity has been the fact that an active adversary who controls a fraction of the participants can attempt to isolate an honest party Anna by simply dropping all of the messages/onions received directly from Anna. In a fault-tolerant network protocol, the remaining participants will still be able to get their messages through to their destinations. An adversary observing who did and did not receive a message will be able to infer who Anna's intended recipient had been.

To resolve this issue, we introduce a new type of onions, called merging onions: a set of merging onions regularizes the number of onions that remain in the system. If released simultaneously, these onions initially move independently from one another and then gradually come to coordinate their movements. When two onions become coordinated, the onions are "merged". This is accomplished by simply dropping one of the two onions. (See Algorithm 2 for details on how to construct merging onions.)

Suppose that Anna embeds her message for Roberto into each onion in a set of merging onions. If the adversary drops many of Anna's onions upfront, then only a few pairs of Anna's onions will merge later on. In this way, Roberto can still receive the same number of messages as he would in the alternative setting where Anna's recipient is Carol instead.

3.1.2 Related work

On definitions There is a diversity of definitions of anonymous channels that have been considered in the literature [BKM+13, BFTS04, CPP08, DRS04, AAC+11]. We consider one that is as strong and natural as possible: the adversary has a complete view of the network and actively controls a constant fraction of the participants; anonymity means that the adversary cannot distinguish two input configurations of its own choosing (but we limit the possibilities to simple I/O, where each participant desires to send a fixed-length message to a unique interlocutor). Backes et al. [BKM+13] assume an adversary with only a partial view of the network. Their notion of anonymity also deviates from standard indistinguishability in that most of the input is selected randomly. Older definitions based on probability and information theory [BFTS04, CPP08, DRS04, AAC+11] assume that the input is random. Encryption schemes that are appropriate for onion routing are known [CL05, BGKM12].

On lower bounds The only prior negative result for onion routing[1] is the Trilemma theorem due to Das et al. [DMMK18], which states that a protocol that realizes anonymous channels has to include $\Omega(N)$ transmissions total (i.e., $\omega(1)$ per party). In contrast, our lower bound is much tighter; it states that $\omega(N \log \lambda)$ communication is required, where additionally λ is the security parameter. To be fair, the Trilemma theorem already applies in the passive adversary setting, whereas our lower bound is for the active adversary setting only.

On protocols Achieving anonymous channels that would satisfy our definition using heavier cryptographic machinery has been considered. One of the earliest examples is Chaum's dining cryptographer's protocol [Cha88a]. Rackoff and Simon [RS93] use multi-party computation (MPC) [GMW87] for providing security from active adversaries. Other cryptographic tools used in constructing anonymity protocols include oblivious RAM (ORAM) and private information retrieval (PIR) [CB95, CBM15].

Other known onion routing protocols are either not anonymous or not fault-tolerant. Vuvuzela [vdHLZZ15], Stadium [TGL+17], Atom[2] (variant #2) [KCDF17] and Π_a [ALU18] were not shown to be anonymous. In fact, using Theorem 6, we can see that they cannot be anonymous since these protocols do not equalize. In Vuvuzela, in Stadium and in Π_a,

[1] The Trilemma theorem applies to a category of communications protocols that can be modeled by pebbling in a petri-net [Jen13] and shares similar abstract traits with onion routing.

[2] Technically, Atom is not an onion routing protocol. However, each message packet in Atom is layered encryption object like an onion.

adding a random (polynomially-bounded) number of dummy onions can provide differential privacy but not anonymity. In Atom #2, the adversary can drop an honest message packet upfront and know that any message received in the end was not sent by the sender of the dropped packet. Vuvuzela, Stadium and Atom (variant #1) [KCDF17] provide anonymity from the active adversary by ensuring that when the adversary drops a single message packet, all parties abort [vdHLZZ15, TGL+17, KCDF17]. Aborting the run from a single lost message packet is highly impractical for use in a real system. Moreover, detecting a single lost message packet can be labor-intensive, e.g., relying on a combination of local broadcasts and verifiable shuffling [TGL+17, KCDF17]. Like the results in this paper, these results (Vuvuzela [vdHLZZ15], Stadium [TGL+17], Atom [KCDF17] and Π_a [ALU18]) are also for the simple I/O setting in the synchronous communication model.

In contrast to all of these, our protocol $\Pi_{\bowtie}$ is the first to be simultaneously anonymous and practical (fault-tolerant, distributed and with low communication complexity). Importantly, it is also the first protocol to achieve anonymity from the corrupted recipient. See Table 3.1.

Protocol	Server load	# of rounds	Fault tolerant?	Security
Vuvuzela [vdHLZZ15]	N	n	no	diff. privacy
Stadium [TGL+17]	$\Theta\left(\frac{N}{\lvert G\rvert}\right)$	$\Theta(\lvert G\rvert)$	no	diff. privacy
Atom [KCDF17]	$\Theta\left(\frac{N\lvert G\rvert}{n}\right)$	$\Theta(\lvert G\rvert)$	#1: no #2: yes	anonymity k-anonymity
Π_a [ALU18]	$\mathsf{polylog}(\lambda)$	$\mathsf{polylog}(\lambda)$	yes	diff. privacy
$\Pi_{\bowtie}$ (this work)	$\mathsf{polylog}(\lambda)$	$\log N\mathsf{polylog}(\lambda)$	yes	anonymity

Table 3.1: A comparison of properties of provably secure onion routing (and similar) protocols. Stadium [TGL+17] and Atom [KCDF17] both rely on groups of servers functioning as nodes in a random permutation network. In the table entries: N is the number of participants; n is the number of servers; and $|G|$ is the size of a group of servers. In general, we want n and $|G|$ to be at least $\mathsf{polylog}(\lambda)$ to ensure that at least one server in each group is honest.

3.2 Preliminaries and modeling the problem

The notation, primitive (i.e., secure onion encryption scheme) and problem setting introduced in Sections 2.2 and 2.3 also pertain to this chapter.

3.2.1 Martingales and the Azuma-Hoeffding bound

The definition of a martingale and the Azuma-Hoeffding inequality [MU05, Theorem 12.4] are standard notions but provided below for convenience.

Definition 9 (Martingale [MU05, Definition 12.1])**.** *A sequence of random variables $Z_0, Z_1, \ldots$ is a martingale if for all $i \geq 0$, the following conditions are satisfied:*

1. $\mathbb{E}[|Z_i|] < \infty$, *and*
2. $\mathbb{E}[Z_{i+1}|Z_0, \ldots, Z_i] = Z_i$.

Theorem 4 (Azuma-Hoeffding inequality [MU05, Theorem 12.4])**.** *Let the sequence of random variables $X_0, X_1, \ldots$ be a martingale, where for every $i > 0$, $|X_i - X_{i-1}| \leq c_i$. Then for any $j > 0$ and $\lambda > 0$, we have that*

$$\Pr[|X_j - X_0| \geq \lambda] \leq e^{-\frac{\lambda^2}{2\sum_{i=1}^{j} c_i}}.$$

See Mitzenmacher and Upfal's book [MU05, Theorem 12.4] for the proof.

3.3 Anonymity, equalizing and mixing

In this section, we provide game-based definitions. Our definition of anonymity is essentially standard indistinguishability, but we also introduce two related definitions: *equalizing* (Definition 11) and *mixing* (Definition 12).

3.3.1 Anonymity

The anonymity game $\mathsf{AnonymityGame}(1^\lambda, \Pi, \mathcal{A}, \Sigma)$ is parametrized by the security parameter 1^λ, a protocol Π, an adversary $\mathcal{A}$, and a set Σ of input vectors.

The adversary $\mathcal{A}$ chooses a subset $\mathsf{Bad} \subseteq [N]$ of the parties to corrupt and also chooses the keys for these parties. Let $\mathsf{pk}(\mathsf{Bad})$ be a shorthand for the corrupted parties' public keys. $\mathcal{A}$ chooses two input vectors $\sigma^0, \sigma^1 \in \Sigma$ such that $\sigma^0 \equiv_{\mathsf{Bad}} \sigma^1$ and sends $(\mathsf{Bad}, \mathsf{pk}(\mathsf{Bad}), \sigma^0, \sigma^1)$ to the challenger $\mathcal{C}$.

For each honest party in $[N] \setminus \mathsf{Bad}$, $\mathcal{C}$ generates a key pair for the party; the public keys $\mathsf{pk}([N] \setminus \mathsf{Bad})$ of the honest parties are sent to $\mathcal{A}$.

The challenger $\mathcal{C}$ chooses a random bit $b \leftarrow_\$ \{0, 1\}$ and interacts with $\mathcal{A}$ in an execution of protocol Π on input σ^b with $\mathcal{C}$ acting as the honest parties adhering to the protocol and $\mathcal{A}$ controlling the corrupted parties.

At the end of the execution, $\mathcal{A}$ computes a guess b' for b from its view $\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^b)$ and wins the anonymity game if $b' = b$. See Figure 3.1.

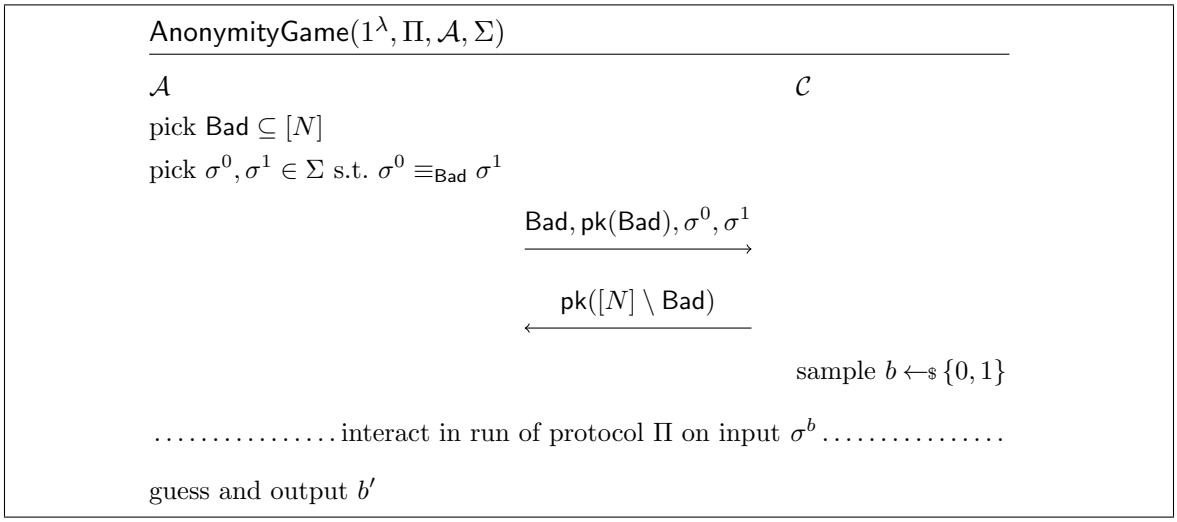

Figure 3.1: Schematic of the anonymity game.

The standard notion of anonymity is

Definition 10 (Anonymity)**.** *A protocol* $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$ *is anonymous from the adversary class* $\mathbb{A}$ *w.r.t. the input set* Σ *if every adversary* $\mathcal{A} \in \mathbb{A}$ *wins the anonymity game* $\mathsf{AnonymityGame}(1^\lambda, \Pi, \mathcal{A}, \Sigma)$ *with only negligible advantage, i.e.,*

$$\left| \Pr\left[\mathcal{A} \textit{ wins } \mathsf{AnonymityGame}(1^\lambda, \Pi, \mathcal{A}, \Sigma) \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda) .$$

The protocol is computationally (resp. statistically) anonymous if the adversaries in $\mathbb{A}$ *are computationally bounded (resp. unbounded).*

3.3.2 Equalizing

We introduce a new definition, called equalizing, which is closely related to anonymity. This is a useful definition to have because in many cases, as we shall see in Theorem 6, equalizing and mixing (defined next) implies anonymity, and proving that a protocol satisfies equalizing and mixing separately, is often easier than proving that the protocol is anonymous without this breakdown.

Equalizing is defined with respect to the equalizing game (below), in which the distinguisher attempts to determine whether the setting is σ^0 or σ^1 from just the sizes of the parties' outputs (and the description of the adversary and σ^0 and σ^1). The equalizing game

$\mathsf{EqualizingGame}(1^\lambda, \Pi, \mathcal{A}, \mathcal{D}, \Sigma)$ is parametrized by the security parameter 1^λ, protocol Π, an adversary $\mathcal{A}$, a distinguisher $\mathcal{D}$ and a set Σ of input vectors.

The game starts exactly like the anonymity game: The adversary $\mathcal{A}$ chooses a subset $\mathsf{Bad} \subseteq [N]$ of the parties to corrupt and also chooses the keys for these parties. Let $\mathsf{pk}(\mathsf{Bad})$ be a shorthand for the corrupted parties' public keys. $\mathcal{A}$ chooses two input vectors $\sigma^0, \sigma^1 \in \Sigma$ such that $\sigma^0 \equiv_{\mathsf{Bad}} \sigma^1$ and sends $(\mathsf{Bad}, \mathsf{pk}(\mathsf{Bad}), \sigma^0, \sigma^1)$ to the challenger $\mathcal{C}$.

For each honest party in $[N] \setminus \mathsf{Bad}$, $\mathcal{C}$ generates a key pair for the party; the public keys $\mathsf{pk}([N] \setminus \mathsf{Bad})$ of the honest parties are sent to $\mathcal{A}$.

The challenger $\mathcal{C}$ chooses a random bit $b \leftarrow_\$ \{0, 1\}$ and interacts with $\mathcal{A}$ in an execution of protocol Π on input σ^b with $\mathcal{C}$ acting as the honest parties adhering to the protocol and $\mathcal{A}$ controlling the corrupted parties.

At the end of the execution, the challenger $\mathcal{C}$ determines the size v_i of the output of each party $i \in [N]$, and sends the statistics $v = v_1, v_2, \ldots, v_N$ to the distinguisher $\mathcal{D}$, along with the description $\mathsf{Desc}(\mathcal{A})$ of the adversary and the adversary's choices σ^0 and σ^1 for the input to the protocol. ($\mathsf{Desc}(\mathcal{A})$ is the program encoding of the UTM $\mathcal{A}$ as understood by a state function; it does not include the randomness of the run.)

The distinguisher $\mathcal{D}$ computes a guess b' for b from $\mathsf{Desc}(\mathcal{A})$, σ^0, σ^1 and v and wins the game if $b' = b$. See Figure 3.2.

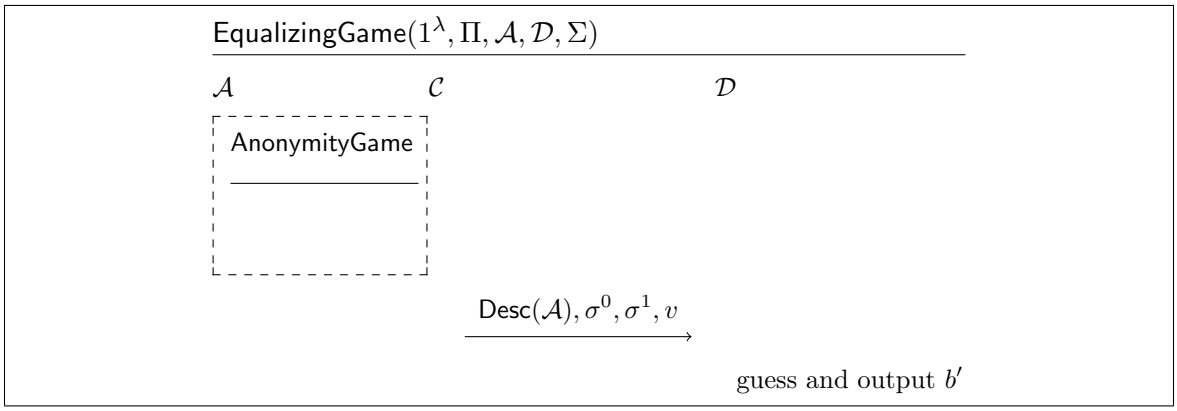

Figure 3.2: Schematic of the equalizing game.

The definition for equalizing is as follows.

Definition 11 (Equalizing)**.** *A protocol* $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$ *equalizes for the adversary class* $\mathbb{A}$ *w.r.t. the input set* Σ *if for every adversary* $\mathcal{A} \in \mathbb{A}$ *and for every distinguisher* $\mathcal{D}$*,* $\mathcal{D}$ *wins* $\mathsf{EqualizingGame}(1^\lambda, \Pi, \mathcal{A}, \mathcal{D}, \Sigma)$ *with negligible advantage, i.e.,*

$$\left| \Pr\left[\mathcal{D} \text{ wins } \mathsf{EqualizingGame}(1^\lambda, \Pi, \mathcal{A}, \mathcal{D}, \Sigma) \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda) \,.$$

The protocol computationally (resp. statistically) equalizes if the distinguisher is computationally bounded (resp. unbounded).

3.3.3 Mixing

Our definition of mixing captures the idea that a protocol doesn't mix if the adversary can trace an honest message to its origin without being informed whether his challenge is valid or not.

Let $\tilde{m} \in \mathcal{M}$ be the special challenge message for the mixing game. Let an onion be a "*valid challenge onion*" if peeling the outermost layer produces the challenge message $\tilde{m}$.

At a high level, the game goes like this: The adversary $\mathcal{A}$ picks a set Bad of corrupted parties, the keys $\mathsf{pk}(\mathsf{Bad})$ for these parties and a set S of challenge senders and specifies the inputs for all non-challenge parties. (The adversary aims to trace a valid challenge onion back to a challenge sender.) After interacting in a run of protocol Π, $\mathcal{A}$ chooses two onions o^0 and o^1. The challenger $\mathcal{C}$ samples a sender $s \leftarrow\!\!\$\, \mathsf{S}$ from S. Let s^0 be the sender of o^0, and let s^1 be the sender of o^1. If $\mathcal{A}$ chooses valid challenge onions, and $\{s\} \subseteq \{s^0, s^1\} \subseteq \mathsf{S}$, then $\mathcal{C}$ sets b to be the bit $\tilde{b}$ such that $s = s^{\tilde{b}}$. Otherwise, $\mathcal{C}$ sets $b \leftarrow\!\!\$\, \{0,1\}$ to be a random bit. After setting b, $\mathcal{C}$ sends the identity s of the randomly selected sender to $\mathcal{A}$. $\mathcal{A}$ wins the game he can guess b.

Note that the adversary can lose even in the case where he knows the sender of a valid challenge onion. This can happen if the challenger picks a sender who is not the sender of either challenge onion o^0 or o^1. At first glance, this may seem odd. However, the point is that if the adversary can trace a valid challenge onion to its origin, then the adversary still wins with nonnegligible advantage.

We now define the game more formally. Let $\mathcal{OE} = (\mathsf{Gen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$ be a secure onion encryption scheme. For the *statistical* notion of mixing, $\mathcal{OE}$ is an ideal onion encryption scheme.

Mixing game The mixing game $\mathsf{MixingGame}(1^\lambda, \Pi, \mathcal{A}, \Sigma, \mathsf{Type})$ is parametrized by the security parameter 1^λ, an onion routing protocol Π, an adversary $\mathcal{A}$, a set Σ of input vectors and a type Type. Type can be either "all" or "bad". When Type is set to all, the adversary aims to show that he knows the sender of a receiver. When Type is set to bad, the adversary aims to show that he knows the sender of a corrupted receiver.

The adversary $\mathcal{A}$ chooses a subset $\mathsf{Bad} \subseteq [N]$ of the parties to corrupt and also chooses the keys for these parties. Let $\mathsf{pk}(\mathsf{Bad})$ be a shorthand for the corrupted parties' public

keys. $\mathcal{A}$ also identifies a set $\mathsf{S} \subseteq [N] \setminus \mathsf{Bad}$ of honest *challenge senders* and a set $\mathsf{R} \subseteq [N]$ of *challenge receivers s.t.* $|\mathsf{R}| = |\mathsf{S}|$.

For any sequence $\tilde{\tau} = (\tilde{\tau}_1, \tilde{\tau}_2, \ldots, \tilde{\tau}_N)$ of sets of message-receiver pairs and for any bijection f from a set $A \subseteq [N]$ of parties to a set $B \subseteq [N]$ of parties, let $\tilde{\tau} \sqcup f = (\tau_1, \tau_2, \ldots, \tau_N)$ where $\tau_i = \tilde{\tau}_i \cup \{(\tilde{m}, f(i))\}$ for every $i \in A$ and $\tau_i = \tilde{\tau}_i$ for every $i \notin A$.

Let a sequence $\tilde{\sigma} = (\tilde{\sigma}_1, \tilde{\sigma}_2, \ldots, \tilde{\sigma}_N)$ of sets of message-receiver pairs be a "*partial input vector for senders* S *and receivers* R" if for any bijection $\mathsf{sendsto} : \mathsf{S} \mapsto \mathsf{R}$ mapping from S to R, the sequence $\sigma = \tilde{\sigma} \sqcup \mathsf{sendsto}$ is in the input set Σ.

Finally, for a partial input vector $\tilde{\sigma}$ for S and R, let $\Sigma_{\tilde{\sigma}}^{\mathsf{S},\mathsf{R}}$ be the set of all input vectors $\sigma = \tilde{\sigma} \sqcup \mathsf{sendsto}$ where $\mathsf{sendsto}$ is a bijection from S to R.

In addition to Bad, $\mathsf{pk}(\mathsf{Bad})$, S and R, $\mathcal{A}$ also picks the partial input vector $\tilde{\sigma}$ and sends $(\mathsf{Bad}, \mathsf{pk}(\mathsf{Bad}), \mathsf{S}, \mathsf{R}, \tilde{\sigma})$ to the challenger $\mathcal{C}$.

For each honest party in $[N] \setminus \mathsf{Bad}$, $\mathcal{C}$ generates a key pair for the party by running the onion encryption scheme's key generating algorithm Gen; the public keys $\mathsf{pk}([N] \setminus \mathsf{Bad})$ of the honest parties are sent to the adversary $\mathcal{A}$.

The challenger $\mathcal{C}$ chooses a random input vector $\sigma \leftarrow\!\!\$\, \Sigma_{\tilde{\sigma}}^{\mathsf{S},\mathsf{R}}$ (i.e., picks destinations for the challenge messages) and interacts with $\mathcal{A}$ in an execution of protocol Π on input σ with $\mathcal{C}$ acting as the honest parties adhering to the protocol and $\mathcal{A}$ controlling the corrupted parties. (Whenever the protocol Π specifies for an onion to be formed or processed, $\mathcal{C}$ runs the onion encryption scheme's onion-forming algorithm $\mathsf{FormOnion}$ or onion-processing algorithm $\mathsf{ProcOnion}$.)

If $\mathsf{Type} = \mathsf{all}$, the set $\mathcal{R}$ of receivers is set to all parties $[N]$. Otherwise (if $\mathsf{Type} = \mathsf{bad}$), $\mathcal{R}$ is set to all corrupted parties Bad.

Let $\mathcal{O}_{\mathcal{R}}$ be the set of honest onions received by the parties in $\mathcal{R} \cap \mathsf{R}$.

At the end of the execution, $\mathcal{A}$ chooses two challenge onions o^0 and o^1 from $\mathcal{O}_{\mathcal{R}}$ and sends (o^0, o^1) to the challenger. Let s^0 be the sender of o^0, and let s^1 be the sender of o^1.

The challenger $\mathcal{C}$ samples a random bit $b \leftarrow\!\!\$\, \{0,1\}$ and a random challenge sender $s \leftarrow\!\!\$\, \mathsf{S}$. If $\mathcal{A}$ chose valid challenge onions, and $\{s\} \subseteq \{s^0, s^1\} \subseteq \mathsf{S}$, then $\mathcal{C}$ sets b to be the bit $\tilde{b}$ such that $s = s^{\tilde{b}}$. Otherwise, $\mathcal{C}$ sets $b \leftarrow\!\!\$\, \{0,1\}$ to be a random bit. After setting b, $\mathcal{C}$ sends the identity s of the randomly selected sender to $\mathcal{A}$.

The adversary $\mathcal{A}$ computes a guess b' for b from its view $\mathsf{V}^{\Pi,\mathcal{A},\mathsf{Bad}}(\sigma^b)$ and the identity s of the random sender and wins if $b' = b$. See Figure 3.3.

We now define mixing and mixing for corrupted receivers.

Definition 12 (Mixing)**.** *An onion routing protocol* $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$ *mixes for*

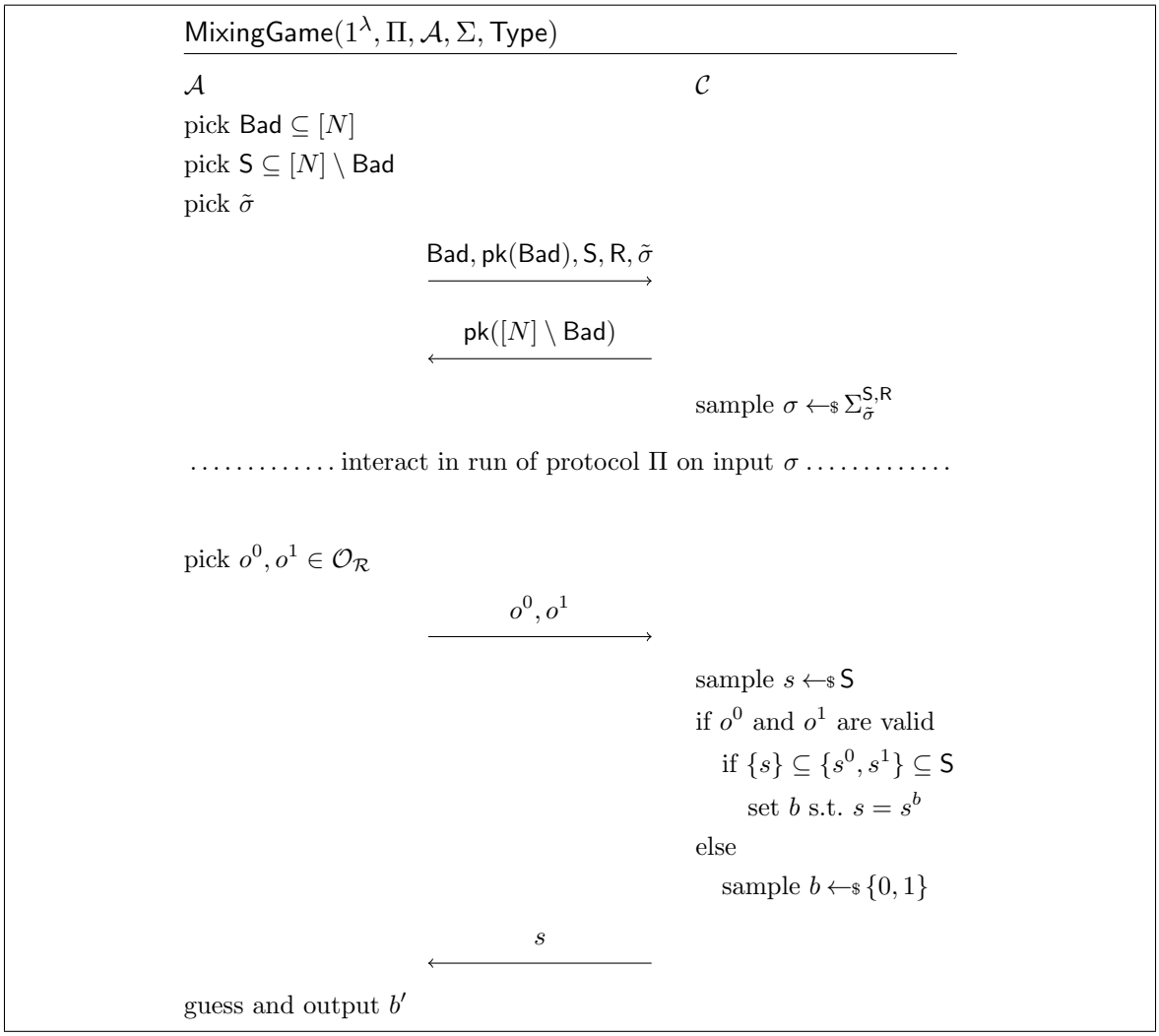

Figure 3.3: Schematic of the mixing game.

the adversary class $\mathbb{A}$ *w.r.t. the input set* Σ *if every adversary* $\mathcal{A} \in \mathbb{A}$ *wins* $\mathsf{MixingGame}(1^\lambda, \Pi, \mathcal{A}, \Sigma, \mathsf{all})$ *with negligible advantage, i.e.,*

$$\left| \Pr\left[\mathcal{A} \textit{ wins } \mathsf{MixingGame}(1^\lambda, \Pi, \mathcal{A}, \Sigma, \mathsf{all}) \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda)\,.$$

The protocol mixes for corrupted receivers if every adversary $\mathcal{A}$ *wins the mixing game with negligible advantage when* Type *is set to "*bad*" instead of "*all*".*

The protocol computationally (resp. statistically) mixes if the adversaries in $\mathbb{A}$ *are computationally bounded (resp. unbounded).*

3.3.4 Relating equalizing and mixing to anonymity

Equalizing and mixing are related to anonymity in the following ways:

Theorem 5. *If the onion routing protocol* Π *is anonymous from the adversary class* $\mathbb{A}_\kappa$ *w.r.t. the input set* Σ *(Definition 10) then* Π *equalizes for* $\mathbb{A}_\kappa$ *w.r.t.* Σ *(Definition 11).*

This is clearly true since the numbers of messages received by the parties are included in the adversarial view. In the next section, we will use Theorem 5 to prove our complexity lower bound (Theorem 7): anonymity requires a superlogarithmic (in the security parameter) onion transmissions per party.

Let an OR protocol be *indifferent* if the destination and nonce of every honest non-message-delivering onion layer are independent of the input.

Theorem 6. *If the indifferent OR protocol* Π *equalizes and mixes for the adversary class* $\mathbb{A}_\kappa$ *w.r.t. the input set* Σ *(Definitions 11 and 12), then* Π *is anonymous for* $\mathbb{A}_\kappa$ *w.r.t.* Σ *(Definition 10).*

We omit the proof of Thereoms 6 for brevity, but it can be found in *Supplementary materials*. In §3.6, we will use Theorem 6 to prove our complexity upper bound (Theorem 8): anonymity is achievable using a polylogarithmic (in the security parameter) onion transmissions per party.

3.3.5 Remarks

On the definition of anonymity While we implicitly define anonymity for the set $[N]$ of participants, at times, it may be desirable to make the set of participants a parameter of anonymity. After all, parties are sometimes offline or online but not sending anything! Fortunately, extending the definition in this way is straightforward.

In the anonymity game, the adversary picks two inputs to the protocol that are equivalent to each other for the adversary's choice Bad of corrupted parties. This is different from the standard "leakage-free" notion of indistinguishability in which the adversary can pick any two inputs. The difference is due to the adversary's ability to look into the corrupted parties. In doing so, the adversary learns the inputs and outputs of the corrupted parties, and so, part of the input to the protocol necessarily leaks. Additionally, the adversary obtains the numbers of honest messages received by the honest parties as part of his view; (see *Outputs and views*).

Suppose that the protocol guarantees the delivery of every honest message. In this case, the adversary receives all honest messages that were instructed to be sent to corrupted parties. This leakage is exactly captured by constraining the adversary's choice for the challenge inputs to an equivalence class defined by Bad. Anonymity means that no information beyond the equivalence class is leaked to the adversary.

Does anonymity imply mixing? In general it does not. Consider the simple protocol in which everyone sends an onion to everyone; for all but the intended recipient the onion is a dummy one. This protocol is anonymous for the network adversary (i.e., without corrupted parties) even though it does not mix.

In a setting where the adversary can corrupt some parties and, therefore, serve as the recipient of some of the messages from honest senders, we can go about creating a reduction. Our reduction will use an adversary that breaks mixing in order to break anonymity. This reduction needs to know the corrupted parties' secret keys in order to play the challenger in the mixing game. So it will go through in a PKI setting where each published public key comes with a zero-knowledge proof of knowledge of the corresponding secret key.

3.4 A lower bound for anonymous OR protocols

Simple I/O setting Let us consider the simplified scenario where everyone sends exactly one message of the same length, and everyone receives exactly one message. Let $\mathsf{SimpleIO}$ be the set of all inputs of the form,

$$\sigma = (\{(m_1, \pi(1))\}, \{(m_2, \pi(2))\}, \ldots, \{(m_N, \pi(N))\}),$$

where $m_1, m_2, \ldots, m_N$ are messages from the message space $\mathcal{M}$, and $\pi : [N] \mapsto [N]$ is a permutation function over the domain $[N]$. We refer to the setting where the input vector σ is constrained to $\mathsf{SimpleIO}$ as *the simple I/O setting*.

For the remainder of the paper, we will be operating in the simple I/O setting. We will generally leave Σ unspecified; by context, we mean $\Sigma = \mathsf{SimpleIO}$.

3.4.1 Anonymity implies polylog onion cost

We prove that, in the simple I/O setting, an OR protocol can be anonymous from the active adversary only if the average number of onions transmitted per party is superlogarithmic in the security parameter.

To prove the lower bound, we make use of the following observation (Lemma 2, below): If an OR protocol Π is too efficient, then there exist many settings in which there exist parties i and k such that i is neither a sender nor an intermediary node for recipient k.

In a run of OR protocol Π interacting with adversary $\mathcal{A}$ on input σ:

- For an honest party i, let $\#\mathsf{onions}^{\Pi,\mathcal{A}}_{i\to j\to k}(\sigma)$ denote the number of onions created by party i and received by party j that will reach party k (if allowed to continue to k).

- For honest parties i and $j \neq i$, "*i cannot affect j's recipient*" if

$$\mathbb{E}\left[\#\mathsf{onions}^{\Pi,\mathcal{A}}_{j\to i\to \mathsf{R}(j)}(\sigma)\right] \leq \frac{1}{2}, \tag{3.1}$$

  where $\mathsf{R}(j)$ is the recipient for j.

Lemma 2. *If the onion cost of the OR protocol Π interacting with the adversary $\mathcal{A}$ is sublinear in the number N of parties, then there exists a set $\mathsf{Inputs} \subseteq \mathsf{SimpleIO}$, $|\mathsf{Inputs}| = \Theta(|\mathsf{SimpleIO}|)$ s.t. for every $\sigma \in \mathsf{Inputs}$, there exists a set $\mathsf{Senders}_\sigma \subseteq [N]$, $|\mathsf{Senders}_\sigma| = \Theta(N)$ s.t. for every party $i \in \mathsf{Senders}_\sigma$,*

i. $\mathbb{E}_{\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma)\right] = \mathcal{O}(1)\cdot \mathsf{OC}^{\Pi,\mathcal{A}}$, *and*

ii. there exists a party $j_{\sigma,i}$ such that i cannot affect $j_{\sigma,i}$'s recipient (as defined in (3.1)).

Proof. From Markov's inequality,

$$\Pr_\sigma\left[\mathbb{E}_{i,\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma)\right] \geq 2\mathsf{OC}^{\Pi,\mathcal{A}}\right] \leq \frac{1}{2}.$$

Thus, there exists a set Inputs, $|\mathsf{Inputs}| = \Theta(|\mathsf{SimpleIO}|)$ s.t. for every $\sigma \in \mathsf{Inputs}$, $\mathbb{E}_{i,\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma)\right] < 2\mathsf{OC}^{\Pi,\mathcal{A}}$. Using Markov's inequality again, we have, for all $\sigma \in \mathsf{Inputs}$,

$$\Pr_i\left[\mathbb{E}_{\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma)\right] \geq 2\mathbb{E}_{i,\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma)\right]\right] \leq \frac{1}{2}.$$

That is, there exists a set $\mathsf{Senders}_\sigma \subseteq [N]$, $|\mathsf{Senders}_\sigma| = \frac{N}{2}$ s.t. for sufficiently large N,

$$\mathbb{E}_{\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma)\right] < 4\mathsf{OC}^{\Pi,\mathcal{A}} < \frac{N}{2}, \qquad \forall i \in \mathsf{Senders}_\sigma. \tag{3.2}$$

This shows that (i) is satisfied.

For every $i \in \mathsf{Senders}_\sigma$, there are at most $N-1$ distinct party $j_{\sigma,i} \neq i$, such that

$$\mathbb{E}\left[\#\mathsf{onions}^{\Pi,\mathcal{A}}_{j_{\sigma,i}\to i\to \mathsf{R}(j_{\sigma,i})}(\sigma)\right] \geq \frac{1}{2},$$

where $\mathsf{R}(j_{\sigma,i})$ is the recipient of $j_{\sigma,i}$ in σ. If this weren't the case, then the expected number of onions that party i transmits would be at least $\frac{N}{2}$, contradicting (3.2). Hence, we also satisfy (ii). □

A communications protocol can be both anonymous and arbitrarily efficient if it does not have to be functional. For example, a protocol in which nothing is ever transmitted achieves anonymity vacuously.

Our lower bound holds for protocols that are minimally functional for the active adversary. We call this notion robustness. **Definition:** Let an OR protocol be *weakly robust*

w.r.t. an input set Σ if for every input $\sigma \in \Sigma$, for every active adversary $\mathcal{A}$ and for every honest party i, $\mathcal{A}$ doesn't drop any onion formed by i implies that all of i's messages are delivered to their respective recipients.

Theorem 7. *For any constant $\kappa > 0$, if the OR protocol $\Pi(1^\lambda, \mathsf{pp}, \mathsf{states}, \$, \sigma)$ is weakly robust and (computationally) anonymous from $\mathbb{A}_\kappa$ then the onion cost of Π interacting with $\mathbb{A}_\kappa$ is superlogarithmic in the security parameter λ.*

For the proof, we show that if the protocol is too efficient, then the adversary can "isolate" an honest party by blocking all network traffic originating from or passing through the party.

Proof. For a party $\ell \in [N]$, let $\mathcal{A}_\ell \in \mathbb{A}_\kappa$ be the adversary who corrupts a uniformly random set of $\lfloor \kappa N \rfloor$ parties and, additionally, drops every onion that ℓ transmits directly to a corrupted party. Otherwise, $\mathcal{A}_\ell$ follows the protocol.

Let $\mathcal{A}$ be the adversary that chooses a random party $\ell \leftarrow\!\!\$\, [N]$ to target and then follows $\mathcal{A}_\ell$'s code.

Assume for the sake of reaching a contradiction that Π is an OR protocol that is weakly robust and anonymous from $\mathbb{A}_\kappa$, and the onion cost of Π interacting with any $\mathcal{A}$ is $\mathcal{O}(\log \lambda)$.

W.l.o.g., assume $\mathcal{O}(\log \lambda) = \mathrm{o}(N)$. We assume this to be the case, since otherwise, there are known solutions with $\Theta\left(N^2\right)$ communication complexity, e.g., using general purpose MPC.

From Lemma 2, there exists an input $\sigma^0 \in \mathsf{SimpleIO}$, such that there exists a set $\mathsf{Senders}_{\sigma^0} \subseteq [N]$, $|\mathsf{Senders}_{\sigma^0}| = \Theta(N)$ s.t. for every party $i \in \mathsf{Senders}_{\sigma^0}$,

i. $\mathbb{E}_{\$}\left[\mathsf{out}_i^{\Pi,\mathcal{A}}(\sigma^0)\right] = \mathcal{O}(\log \lambda)$, and

ii. there exists a party $j_{\sigma^0,i}$ such that i cannot affect $j_{\sigma^0,i}$'s recipient.

We will now prove the following: In the event (with nonnegligible probability) that $\mathcal{A}$ picks a party $i \in \mathsf{Senders}_{\sigma^0}$ to target, $\mathcal{A}$ can distinguish the setting on input σ^0 from the setting on input $\sigma^1 = \mathsf{swap}(\sigma^0, i, j_{\sigma^0,i})$ which is the same as σ^0 except that the inputs for parties i and $j_{\sigma^0,i}$ are swapped.

Let R be the recipient of $j_{\sigma,i}$ in σ^0 (and also the recipient of i in σ^1), and let v_{R}^b denote the number of messages that R receives in a protocol run of Π interacting with adversary $\mathcal{A}$ on input σ^b.

Let $\mathsf{isolated}$ denote the event that $\mathcal{A}$ manages to drop every onion that i transmits.

On input σ^1: Conditioned on $\mathsf{isolated}$, R never receives his message, i.e.,

$$\Pr\left[v^1_{\mathsf{R}}|_{\mathsf{isolated}} = 0\right] = 1. \tag{3.3}$$

On input σ^0: Let $\mathsf{unaffectable}$ denote the event that $\#\mathsf{onions}^{\Pi,\mathcal{A}}_{j_{\sigma^0,i}\to i\to \mathsf{R}}(\sigma^0) = 0$. From (ii), $\Pr[\mathsf{unaffectable}] \leq \frac{1}{2}$. Combined with weak robustness, it follows that R receives his message with nonnegligible probability, i.e.,

$$\Pr\left[v^0_{\mathsf{R}}|_{\mathsf{isolated}} > 0\right] = \mathsf{nonnegl}(\lambda). \tag{3.4}$$

If $\mathsf{isolated}$ occurs with nonnegligible probability on input σ^0: Then, from combining (3.3) and (3.4), Π doesn't equalize; and from Theorem 5, Π is not anonymous.

To complete our proof, it suffices to prove that the probability of $\mathsf{isolated}$ is nonnegligible: From (i), $[\mathsf{out}^{\Pi,\mathcal{A}}_i(\sigma^0)] = \mathcal{O}(\log\lambda)$. From Markov's inequality, there exists a constant $\alpha > 0$, such that $\mathsf{out}^{\Pi,\mathcal{A}}_i(\sigma^0) \leq \alpha\log\lambda$ with nonnegligible probability.

Let $\mathsf{droppable}$ denote the event that $\mathsf{out}^{\Pi,\mathcal{A}}_i(\sigma^0) \leq \alpha\log\lambda$, and let $\mathsf{isolated}|_{\mathsf{droppable}}$ denote $\mathsf{isolated}$ conditioned on $\mathsf{droppable}$.

The probability of $\mathsf{isolated}|_{\mathsf{droppable}}$ is smallest when the location of each of the (at most) $\alpha\log\lambda$ onions that i transmits goes to a different location. This probability is bounded by the probability p that a random $(\alpha\log\lambda)$-size sample from a set of N balls, κN of them which are green, are all green. When $\alpha\log\lambda \leq \sqrt{N}$,

$$p = \frac{\binom{\kappa N}{\alpha\log\lambda}}{\binom{N}{\alpha\log\lambda}} = (1+\mathrm{o}(1))\frac{(\kappa N)^{\alpha\log\lambda}}{(\alpha\log\lambda)!}\cdot(1+\mathrm{o}(1))\frac{(\alpha\log\lambda)!}{N^{\alpha\log\lambda}} = \Theta\left(\kappa^{\alpha\log\lambda}\right),$$

which is nonnegligible in λ. Thus, $\Pr[\mathsf{isolated}\wedge\mathsf{droppable}] = \Pr[\mathsf{droppable}]\cdot\Pr[\mathsf{isolated}|_{\mathsf{droppable}}]$ is nonnegligible in λ. It follows that $\mathsf{isolated}$ occurs with nonnegligible probability. □

3.4.2 Remarks

On the lower bound **Definition:** Let an OR protocol be *robust* w.r.t. an input set Σ if for every input $\sigma\in\Sigma$ and for every active adversary $\mathcal{A}$, $\mathcal{A}$ drops at most a logarithmic (in the security parameter) number of onions implies that w.o.p., all honest messages are delivered. We can also prove the weaker result that if an OR protocol is *robust* and anonymous, then its onion cost is superlogarithmic (in the security parameter). The proof is a simpler contradiction showing that an OR protocol with logarithmic (in the security parameter) onion cost cannot be robust (rather than anonymous).

To prove the lower bound, we used the fact that the adversary knows the number of messages received by each honest party in the protocol run. However, the bound holds even when we exclude these statistics from the adversarial view. We can prove the stronger result by using in place of Theorem 5: If an OR protocol is anonymous from adversaries who corrupt up to $\kappa N + 1$ parties, then it essentially equalizes for adversaries who corrupt up to κN parties.

3.5 An optimally efficient, anonymous OR protocol

Recall that an OR protocol is *robust* w.r.t. an input set Σ if for every input $\sigma \in \Sigma$ and for every active adversary $\mathcal{A}$, $\mathcal{A}$ drops at most a logarithmic (in the security parameter) number of onions implies that w.o.p., all honest messages are delivered. Previously (in Theorem 7), we showed that for an OR protocol to be robust and anonymous, the onion cost must be superlogarithmic in the security parameter. We now claim the matching upper bound:

Theorem 8. *For any constants $\kappa < \frac{1}{2}$ and $\gamma_1, \gamma_2 > 0$, there exists an OR protocol that is robust and (computationally) anonymous from $\mathbb{A}_\kappa$ with onion cost at most $\gamma_1 \log N \log^{3+\gamma_2} \lambda$ where N is the number of parties and λ, the security parameter.*

We prove this by constructing an OR protocol $\Pi_{\bowtie}$ (pronounced *pi-butterfly*) and showing that it has the relevant properties.

We first present a stepping stone construction $\Pi_{\triangle}$ (pronounced *pi-tree*) that showcases a new technique for equalizing; this protocol is robust and anonymous, but it is not efficient. In §3.5.2, we extend $\Pi_{\triangle}$ to an (almost) optimally efficient construction $\Pi_{\bowtie}$.

3.5.1 The stepping stone construction, $\Pi_{\triangle}$

We use a secure onion encryption scheme $\mathcal{OE} = (\mathsf{Gen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$ as a primitive building block (see §2.2 for a description of OE schemes). For every party i, let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ be i's key pair generated from running the key generating algorithm Gen. For every pair (i, j) of parties, let $\mathsf{sk}_{i,j}$ be the shared key known by only i and j.[3]

Onion-forming phase.

During the onion-forming phase, each honest party i creates two types of onions: (1) checkpoint onions and (2) merging onions. Each checkpoint onion is formed from a piece of data

[3]These shared keys do not need to be set up in advance; it is known that they can be generated as needed from an existing PKI, e.g., using Diffie-Hellman.

called a checkpoint datum. (See *Checkpoint data*, *Checkpoint onions* and *Merging onions* for descriptions of these objects and how they are created.)

Every honest party generates an expected χ checkpoint data from which the party forms the checkpoint onions. Additionally, for every message-recipient pair (m, j) in the party's input, the party forms a set of χ merging onions using the message m and the recipient j. All onions are created during the onion-forming phase and released simultaneously in the first round of the execution phase.

Checkpoint data Protocol $\Pi_{\bowtie}$ runs in h epochs, and each epoch lasts d rounds. The last round of each epoch is called a *diagnostic round*. Let the rounds of the protocols be $1, 2, \ldots, hd$, and so the diagnostic rounds are $d, 2d, \ldots, hd$. Each checkpoint onion has a checkpoint (i.e., a pseudorandom nonce) embedded in one of its diagnostic-round layers (i.e., the layers that are meant to be processed after rounds $d, 2d, \ldots, hd$).

A checkpoint datum (ℓ, k, c) is a triple consisting of an index ℓ of a diagnostic-layer round ℓd, a party k and a checkpoint c. To generate an expected χ checkpoint data, each party i runs $\mathsf{GenCkptData}$ (Algorithm 1) below.

Algorithm 1: $\mathsf{GenCkptData}$

Initiate $\mathsf{Ckpts} = \emptyset$ to the empty set.
for *every* $(\ell, k) \in [h] \times [N]$ **do**
    Compute $z = b(F(\mathsf{sk}_{i,k}, \ell||0))$, where $F(\mathsf{sk}_{i,k}, \cdot)$ is a PRF keyed with shared key $\mathsf{sk}_{i,k}$, and $b(\cdot)$ is a binary function such that the output of $b(F(\cdot, \cdot))$ is one with frequency χ/Nh.
    if $z = 1$ **then**
        Set $c = F(\mathsf{sk}_{i,k}, \ell||1)$.
        Add the checkpoint datum (ℓ, k, c) to the set Ckpts.
Return Ckpts.

Checkpoint onions Checkpoint onions are dummies carrying the empty message "$\perp$" for a random recipient [ALU18] and are used for inferring how many honest onions remain in the system.

For every checkpoint datum $(\ell, k, c) \in \mathsf{Ckpts}$, party i first picks hd nodes $p_1, \ldots, p_{\ell d-1}, p_{\ell d+1}, \ldots, p_{hd}, j \leftarrow\!\!\$\, [N]$ for the routing path and $hd - 1$ nonces $s_1, \ldots, s_{\ell d-1}, s_{\ell d+1}, \ldots, s_{hd} \leftarrow\!\!\$\, \mathcal{S}$, all independently and uniformly at random, and forms an onion $o \leftarrow \mathsf{FormOnion}(\perp, p, \mathsf{pk}(p), s)$ from the empty message "$\perp$", the routing path $p = (p_1, \ldots, p_{\ell d-1}, k, p_{\ell d+1}, \ldots, p_{hd}, j)$, the public keys associated with p and the

sequence $s = (s_1, \ldots, s_{\ell d-1}, c, s_{\ell d+1}, \ldots, s_{hd})$ of nonces.

Merging onions Merging onions are message-bearing; a set of *merging* onions carry a message m for a recipient j. The routing paths for a set of merging onions are structured so that pairs of merging onions are designed to meet at the same location and round and *merge*; in actuality, this is accomplished by simply dropping one of the two onions before the next round.

A set of $\chi = 2^{h-1}$ merging onions is formed using a binary tree graph $G_\triangle$ with χ leaves. To form a set of merging onions, party i assigns a random sequence of d parties and a random sequence of d nonces to each node of $G_\triangle$. Each merging onion o^L corresponds to a leaf node x^L; the sequence of relay nodes for o^L is the concatenation of the sequences of parties along the direct path from x^L to the root of $G_\triangle$, and the sequence of nonces for o^L is the concatenation of the sequences of nonces along the direct path from x^L to the root of $G_\triangle$. See FormMergingOnions (Algorithm 2).

Algorithm 2: FormMergingOnions

Construct a binary tree $G_\triangle$ with $\chi = 2^{h-1}$ leaves.
Label the root of $G_\triangle$: x^0.
for *every labeled node* x^B *with unlabeled children* **do**
    Label the left-child of x^B: $x^{0||B}$ and the right-child of x^B: $x^{1||B}$.
for *every node* x^B **do**
    Assign d randomly selected parties $q^B = q_1^B, q_2^B \ldots, q_d^B \leftarrow_\$ [N]$ and d randomly selected nonces $t^B = t_1^B, t_2^B, \ldots, t_d^B \leftarrow_\$ \mathcal{S}$.
Let $x^{L_1}, x^{L_2}, \ldots, x^{L_{2^{h-1}}}$ be the leaf nodes in $G_\triangle$; each leaf node x^L corresponds to an onion o^L.
for *each leaf node* $x^{L=b_1,b_2,\ldots,b_{h-1}}$ **do**
    Set the sequence p of intermediary parties for o^L to be the concatenation of the sequences of parties of the nodes along the direct path from x^L to the root x^0, i.e., $\rho = q^{b_1,b_2,\ldots,b_{h-1}}||q^{b_2,b_3,\ldots,b_{h-1}}||\ldots||q^{b_{h-1}}$.
    Set the routing path p of the onion o^L to be the concatenation of the intermediaries ρ and the recipient j, i.e., $p = \rho||j$.
    Set the sequence s of nonces for o^L to be the concatenation of the sequences of nonces of the nodes along the direct path from x^L to x^0, i.e., $s = t^{b_1,b_2,\ldots,b_{h-1}}||t^{b_2,b_3,\ldots,b_{h-1}}||\ldots||t^{b_{h-1}}$.
    Form onion $o^L \leftarrow \mathsf{FormOnion}(m, p, \mathsf{pk}(p), s)$ by running FormOnion on the message m, the routing path p, the public keys $\mathsf{pk}(p)$ corresponding to p and the nonces s.
Return $o^1, o^2, \ldots, o^\chi$.

Execution phase After each round, each honest party merges onions with the same nonce value. After each diagnostic round, each honest party runs a diagnostic test; (see *Diagnostic test* below). Depending on the outcome of the diagnostic test, the party either *aborts* (stop routing onions from other parties) or sends the (remaining) processed onions to their next destinations in random order in the next round.

Diagnostic test By properties of secure encryption [Can01, CL05], the adversary cannot meaningfully alter honest onions; so her choices for what to do with an honest onion are limited to just two options: forward the onion to its next destination or drop it. Any alteration to the onion is equivalent to dropping it altogether.

The diagnostic test ensures that the adversary cannot drop too many onions without the honest parties noticing. After every diagnostic round ℓd, party i occasionally retrieves a checkpoint from processing an onion. Party i can verify that the checkpoint is *genuine* by checking it against a list of nonces that i expects to see at this point. This list contains precisely the checkpoints that i embedded into the ℓd-th layers of checkpoint onions for other parties to verify at at round ℓd. If i counts a sufficiently large number of genuine checkpoints, this indicates that the adversary hasn't dropped too many onions, and so it is safe to proceed with the execution.

Let N be the number of parties, and let λ be the security parameter. For any small constant $\epsilon > 0$, $\Pi_{\triangle}$ is anonymous when $\chi = \Omega\left(2^{\lceil \log(A(\log A+1))\rceil}\right)$, where $A = \max(\sqrt{N \log^{2+\epsilon} \lambda}, \log^{2(1+\epsilon)} \lambda)$. See Appendix 3.6.4 in *Supplementary materials* for a full analysis.

3.5.2 The main construction, $\Pi_{\bowtie}$

We now present an extension of $\Pi_{\triangle}$, called $\Pi_{\bowtie}$, that decreases the onion cost in exchange for more rounds.

Like $\Pi_{\triangle}$, $\Pi_{\bowtie}$ consists of an onion-forming phase and an execution phase. All onions are formed during the onion-forming phase (each honest party forms χ merging onions per message-recipient pair in her input and expected χ checkpoint onions) and released simultaneously at the start of the execution phase. The execution phase progresses in epochs, each epoch lasting d rounds. $\Pi_{\bowtie}$ is essentially the same as $\Pi_{\triangle}$ with a *mixing phase*, lasting L epochs, replacing the first epoch of $\Pi_{\triangle}$. We refer to the rest of the execution phase as the *merging phase*.

In $\Pi_{\triangle}$, each party must send many (more than $\sqrt{N} \cdot \mathbf{polylog}(\lambda)$ where N is the number of parties, and λ is the security parameter) onions to ensure that the party's onions spread

out rapidly (in a couple of rounds) across the network. In this way, the adversary cannot drop all of an honest party's onions without also dropping many other dummy onions and enabling the honest parties to detect that an attack is underway.

By mixing the onions in smaller batches, it becomes easier to detect that the adversary is dropping onions. This translates into requiring fewer onions. In $\Pi_{\bowtie}$, we do this by restricting the communication at each epoch of the mixing phase to butterfly network connections; see *Onion-forming phase* below.

Onion-forming phase W.l.o.g., assume that the number $N = 2^{n-1}$ of parties is a power of two; and let $n \stackrel{\text{def}}{=} \log N + 1$.

Let G be the butterfly network such that the switching nodes at every stage are the N parties. Let b^i denote the binary representation of $i-1$. For every stage $\tau \in [n]$ of G, we say that the set $\{i, j\}$ of parties is a "*subnet at stage* τ" if b^i is b^j with the τ-th bit flipped, i.e., if $b^i = b^j_1 \dots b^j_{\tau-1} \overline{b^j_\tau} b^j_{\tau+1} \dots b^j_{n-1}$. We denote by $\mathcal{P}^i_\tau$, the subnet at τ that contains i.

Let G^D be the iterated butterfly network that is D iterations of G.

To convert the algorithm in $\Pi_{\triangle}$ for choosing an onion's routing path to the corresponding algorithm in $\Pi_{\bowtie}$, the sender first picks a random route $w_1, w_2, \dots, w_L$ through G^D, where $L = nD$ is the number of epochs in the mixing phase. Next, for each w_τ, the sender picks $d-1$ parties $q^\tau_1, q^\tau_2, \dots, q^\tau_{d-1}$ such that each party is chosen independently and uniformly at random from the subnet $\mathcal{P}^{w_\tau}_{\tau'}$, where $\tau' = \tau \% n$ denotes the remainder of τ divided by n. Then the routing path P of O is set to the concatenation of $(q^1 || q^2 || \dots || q^L)$ and $(p_{d+1}, p_{d+2}, \dots, p_{hd+1})$, where for each τ, $q^\tau = (q^\tau_1, q^\tau_2, \dots, q^\tau_{d-1}, w_\tau)$, and $(p_1, p_2, \dots, p_{hd+1})$ is the routing path of o. See ConvertOnion (Algorithm 3).

Execution phase During the execution phase, each (unaborted) honest party processes onions, tallies checkpoints and merges onions as needed, and either aborts the protocol or sends the remaining processed onions to their next destinations.

Let $\kappa < \frac{1}{2}$ be the upper bound on the fraction of parties that can be corrupted. For each diagnostic round $\ell d \in \{d, 2d, \dots, (L+h)d\}$, the party aborts the protocol run if there are more than $\frac{W}{3}$ missing checkpoints, where $W = (1-\kappa)\frac{\chi}{nD+h}$ is the expected number of checkpoints.

Let an "*abort message*" be an onion formed using the special abort message, the routing path consisting of just the recipient, the public key of the recipient and the empty nonce. If an honest party aborts, then for every round after, the party sends out χ abort messages to a random sample of parties. An unaborted honest party aborts if he receives an abort

Algorithm 3: ConvertOnion

Let $L = nD$.
Pick a random walk $w_1, w_2, \ldots, w_L$ through G^D. (To embed a checkpoint info (ℓ, k, c) into the onion, set w_ℓ to k and randomly walk backwards from w_ℓ to w_1 and forwards from w_ℓ to w_L.)
for *each epoch* $\tau \in [L]$ **do**
    Pick $d-1$ parties $q_1^\tau, q_2^\tau, \ldots, q_{d-1}^\tau \leftarrow\!\!\$\, \mathcal{P}_{\tau'}^{w_\tau}$ independently and uniformly at random from $\mathcal{P}_\tau^{w_\tau}$. Set $q^\tau = q_1^\tau, q_2^\tau, \ldots, q_{d-1}^\tau, w_\tau$, where $\tau' = \tau \% n$.
Set $q = q^1 || q^2 || \ldots || q^L$.
Set the routing path $P = q||(p_{d+1}, p_{d+2}, \ldots, p_{hd+1})$ of O to be the concatenation of q and $(p_{d+1}, p_{d+2}, \ldots, p_{hd+1})$ where $p = (p_1, p_2, \ldots, p_{hd+1})$ is the routing path for o.
Set the subsequence $t = t_1, t_2, \ldots t_{Ld}$ of nonces (for the mixing phase) to be the length-Ld all-empty-nonce vector. (To embed the checkpoint info (ℓ, k, c), set $t_{\ell d}$ to c and the remaining $Ld - 1$ nonces to the empty nonce.)
Set the sequence $S = t||(s_{d+1}, s_{d+2}, \ldots, s_{hd})$ of nonces for O to be the concatenation of t and $(s_{d+1}, s_{d+2}, \ldots, s_{hd})$ where $s = (s_1, s_2, \ldots, s_{hd})$ is the sequence of nonces for o.
Form onion O using the message for o, the routing path P, the public keys $\mathsf{pk}(P)$ corresponding to P, and the sequence S of nonces.
Return O.

message.

3.6 Proof that $\Pi_{\bowtie}$ is anonymous

For our analysis, effectively working in Canetti's $\mathcal{F}_{\mathsf{Enc}}$-hybrid model [Can01], we assume that each party has associated with it a public key generated using an ideal onion routing scheme, and that F in Algorithm 1 is truly random; it is well understood how to relate this to security in the standard model.

3.6.1 Proof idea

Suppose that the following claim holds:

Theorem 9. *For any constants* $\kappa < \frac{1}{2}$ *and* $\epsilon_1, \epsilon_2 > 0$, $\Pi_{\bowtie}$ *equalizes for* $\mathbb{A}_\kappa$ *when* $\chi = d = D = \epsilon_1 \log^{1+\epsilon_2} \lambda$.

Then, we can prove Theorem 8 as follows.

Proof of Theorem 8. $\Pi_{\bowtie}$ is clearly robust; i.e., $\mathcal{A}$ drops at most a logarithmic (in the security parameter) number of onions implies that w.o.p., all honest messages are delivered.

Recall that an OR protocol is *indifferent* if the destination and nonce of every honest non-message-delivering onion layer are independent of the input. From Theorem 6, $\Pi_{\bowtie}$ is anonymous since it is indifferent, and it mixes [ALU18, Theorem 11 and Lemma 12] and equalizes (Theorem 9).

The onion cost is bounded by the product of the maximum number X of onions formed per honest party and the number of rounds in a full (unaborted) execution of the protocol. With overwhelming probability,

$$\begin{aligned}
&X \cdot (\#\text{ of rounds per epoch}) \cdot (\#\text{ of epochs}) \\
&= X \cdot d \cdot ((\#\text{ of epochs for mixing}) + (\#\text{ of epochs for merging})) \\
&\leq 3\chi(\epsilon_1 \log^{1+\epsilon_2} \lambda)((\log N + 1)(\epsilon_1 \log^{1+\epsilon_2} \lambda) + \log \chi) \qquad (3.5) \\
&\leq 6\epsilon_1^3 \log N \log^{3(1+\epsilon)} \lambda, \qquad (3.6)
\end{aligned}$$

where (3.5) follows since $X \leq 3\chi$ from Chernoff bounds, and (3.6) follows from the assumption that $N = \omega(\log \lambda)$. (We assume this to be the case, since otherwise, there are known solutions with $\Theta\left(N^2\right)$ communication complexity.) We obtain the desired result by setting $\gamma_1 = 6\epsilon_1^3$ and $\gamma_2 = 3\epsilon_2$. □

To complete our proof of Theorem 8, we now prove Theorem 9 (above).

A checkpoint onion reveals its *origin* (the party who formed the onion) to the intermediary node who verifies its checkpoint. If the intermediary node is corrupted, then the adversary learns the origin. So, rather than analyzing what happens to all honest onions, we will analyze what happens to indistinguishable onions, where an *indistinguishable* onion is either (1) a merging onion formed by an honest party or (2) a checkpoint onion formed by an honest party for verification by an honest party. (We don't need to analyze onions that are not indistinguishable since they don't reveal anything about the input and don't affect indistinguishable onions.)

Attack 1: targeting a sender At the start of the execution phase, the adversary knows the origin of every onion. As the execution progresses, the adversary loses this information. But before this occurs, the adversary can attempt to eliminate all of a target sender's indistinguishable onions. If the adversary can successfully do this, then $\Pi_{\bowtie}$ does not equalize (and is, therefore, not anonymous).

In Lemma 3, we show that if an honest party is unaborted at any round of the merging phase, then w.o.p., a constant fraction of every honest party's onions remained in the system at the start of the merging phase. The implication of this is that the adversary

cannot successfully "isolate" an honest sender from the rest of the network by dropping all her onions upfront.

Attack 2: dropping indistinguishable onions Let a pair of indistinguishable onions be *mergeable* if the onions arrive at the same place and time and produce the same nonce when processed (once). Let an indistinguishable onion be a *singleton* if it does not belong in any mergeable pair.

At every round, the adversary observes some statistics on singletons and pairs of mergeable onions. With overwhelming probability, these are the only categories of indistinguishable onions in the system; e.g., w.o.p., there cannot be three onions that produce the same nonce when peeled (once). This is because a pair of mergeable onions at the start of an epoch cannot remain unmerged for too long. Either the adversary drops or merges the pair; or w.o.p., within the epoch (lasting a polylogarithmic number of rounds), an honest party merges the pair. This last fact follows from Chernoff bounds.

By the start of the merging phase, the adversary no longer knows the origin of any indistinguishable onion. However, the adversary may know that fewer merging onions from one honest party, Anna, remain in the system compared with those from another honest party, Allison; in which case, the adversary might bet that more singletons are Anna's merging onions than Allison's. Thus, during the merging phase, the adversary may drop singletons in an attempt to prevent the protocol from equalizing.

In Lemma 6, we prove that the adversary cannot prevent the protocol from equalizing by dropping only singletons.

Of course, the adversary can also drop mergeable pairs. Thus, to conclude our proof of Theorem 9, we show the protocol equalizes when the adversary also drops mergeable pairs.

3.6.2 Subverting attack 1

We now state and prove Lemma 3.

Let $L \stackrel{\text{def}}{=} nD$ be the number of epochs in the mixing phase, and let "*the start of the merging phase*" be the L-th diagnostic round.

Lemma 3. *In a protocol run of $\Pi_{\bowtie}$ interacting with an adversary who corrupts up to $\kappa < \frac{1}{2}$ parties, for every party i, let V^i denote the number of i's merging onions at the start of the merging phase. If there exists an unaborted honest party after the L-th diagnostic, then w.o.p., for all honest i, $V^i \geq \frac{(1-\kappa)\chi}{3}$.*

Proof. Let $\mathcal{A}$ be the adversary who corrupts the maximum number $\lfloor \kappa N \rfloor$ of parties and "targets" an honest party i. After round 1, $\mathcal{A}$ drops every onion that i sends to a corrupted party at round 1. After round 2, $\mathcal{A}$ drops every onion that could have been formed by i (from $\mathcal{A}$'s perspective) that goes to a corrupted party at round 2. Otherwise, $\mathcal{A}$ follows the protocol.

For any arbitrarily small positive constant $\delta > 0$, at least $(1-\delta)(1-\kappa)\chi$ of i's merging onions go to an honest party at round 1 (Chernoff bounds), and at least $\frac{1-\delta}{2}$ fraction of these go to an honest party in round 2 (Chernoff bounds). For any $\delta \leq 1 - \sqrt{\frac{2}{3}}$, if the protocol run were to continue, w.o.p., at least $\frac{(1-\kappa)\chi}{3}$ of i's merging onions would remain at the start of the merging phase.

Let a subnet be *so-so* if it consists of an honest party and a corrupted party.

W.l.o.g., at least one of i's onion is received by an honest party in a so-so subset at round 1. (Otherwise, i's onions would mix at *good* subnets, each consisting of two honest parties.) Thus, using a concentration bound for the hypergeometric distribution [HS05], w.o.p., $\mathcal{A}$ causes at least one honest party to abort at the first diagnostic. This causes all honest parties to eventually abort via abort messages. (Until half of the honest parties abort, the number of aborted honest parties grows faster than exponentially w.r.t. the number of rounds. This follows from recasting the problem as a martingale problem and applying the Azuma-Hoeffding inequality; see Lemma 10 in *Supplementary materials*.)

Any adversary that drops at least as many onions as $\mathcal{A}$ causes the honest parties to abort.

Otherwise, if the adversary deviates from $\mathcal{A}$ either by dropping fewer onions or waiting to drop onions, then (in an unaborted execution) at least $\frac{(1-\kappa)\chi}{3}$ of i's merging onions would remain at the start of the merging phase. □

3.6.3 Subverting attack 2

Let $\mathcal{I}_\ell$ be the set of indistinguishable singletons in the first round of the ℓ-th epoch, and let Y be the (total) number of indistinguishable checkpoint onions that are formed.

Suppose that for every epoch ℓ, the adversary drops α_ℓ fraction of the onions in $\mathcal{I}_\ell$. We expect that the adversary drops α_1 of the Y indistinguishable checkpoint onions during epoch 1, and another α_2 of the remaining $(1-\alpha_1)Y$ indistinguishable checkpoint onions during epoch 2, and so on. Following this logic, *by* the ℓ-th epoch, we expect that $\mathbb{E}[\zeta_\ell]$ fraction of the Y indistinguishable checkpoint onions have been dropped, where $\mathbb{E}[\zeta_\ell]$ is

defined recursively as follows:

$$\mathbb{E}[\zeta_1] = 0 \tag{3.7}$$

$$\mathbb{E}[\zeta_\ell] = \sum_{\tau=1}^{\ell-1}(1 - \mathbb{E}[\zeta_\tau])\alpha_\tau, \qquad \ell \geq 2. \tag{3.8}$$

From repeated applications of probability concentration bounds (see *Supplementary materials*), we can show that (1) the adversary essentially drops a random sample from the remaining singletons (Lemma 7), (2) w.o.p., the actual fraction ζ of dropped indistinguishable checkpoint onions is close to $\mathbb{E}[\zeta]$ (Lemma 9), and (3) w.o.p., the number of missing checkpoint onions at a party and round is strongly correlated with ζ (Lemma 8). It follows that

Lemma 4. *In $\Pi_{\bowtie}$: For every epoch ℓ, let α_ℓ be the fraction of remaining indistinguishable singletons that the adversary drops during the ℓ-th epoch, and let $\mathbb{E}[\zeta_\ell]$ be as defined by* (3.7) *and* (3.8)*.*

For every epoch $L < \ell \leq R$ (in the merging phase),

i If $\mathbb{E}[\zeta_\ell] \geq \frac{1}{2}$, then w.o.p., every honest party aborts by the ℓ-th diagnostic.

ii (Conversely, if there is an unaborted honest party after the ℓ-th diagnostic, then w.o.p., at least half of the indistinguishable checkpoint onions remain in the system at the ℓ-th diagnostic round.)

From Lemma 3, a constant fraction $V^i/\chi > 0$ of each honest party i's merging onions remains in the system at the start of the merging phase. However, the numbers (the V's) of merging onions may be different.

If the adversary were passive, onions would merge for the first time at the start of the $(L+1)$-st epoch, in which case, at the end of the $(L+1)$-st epoch, $U_{L+1} \stackrel{\text{def}}{=} \frac{1}{2}\chi$ of every party's merging onions would remain in the system. Following this trend, at the end of the ℓ-th epoch, $U_\ell \stackrel{\text{def}}{=} \frac{1}{2^{\ell-L}}\chi$ of every party's merging onions would remain in the system.

For our setting (when the adversary is active): Let V_ℓ^i denote the number of i's merging onions at the ℓ-th diagnostic round. To prove equalizing, we want that there exists an epoch ℓ such that for every pair of honest parties i and j, the quantities V_ℓ^i and V_ℓ^j are statistically close. In Lemma 6, we will first show this to be the case when the adversary drops only singletons.

To prove Lemma 6, we make use of Lemma 5, below; (the proof of which follows from casting the problem as a martingale problem and applying the Azuma-Hoeffding inequality; see *Supplementary materials*).

Lemma 5. *Let $\mathcal{U}$ be a set of $2u$ balls paired into $u = \mathsf{polylog}(\lambda)$ distinct pairs of balls, and let $\mathcal{V}$ be a random subset of $\mathcal{U}$, such that $\nu = |\mathcal{V}|/|\mathcal{U}|$ is a constant factor. For any constant $\frac{2v-\nu}{2v-1} - 1 < \delta \leq 1$, w.o.p. in λ, the number W of paired balls in $\mathcal{V}$ is at least $(1-\delta)\nu|\mathcal{V}|$ and at most $(1+\delta)\nu|\mathcal{V}|$.*

A consequence of Lemma 5 is that w.o.p., the number of an honest party's *mergeable pairs* at the start of any epoch is close to what is expected given the number of the party's merging onions at the start of the epoch.

Let $\epsilon > 0$ be any small constant such that $(1+\epsilon)/(1-\epsilon) \leq 1 + \epsilon_2$. Let the "*partway point*" be the R-th diagnostic round, where $R \stackrel{\text{def}}{=} L + \lceil h/\epsilon \rceil$. (Before the partway point, the expected number of indistinguishable merging onions per party and round is polylogarithmic in the security parameter.) We now state and prove Lemma 6.

Lemma 6. *In $\Pi_{\bowtie}$: If between the start of the merging phase and the partway point, the adversary drops only singletons, then w.o.p., for any two honest parties i and j, the number V^i_{L+h} of i's merging onions at the end of the execution is equal to the number V^j_{L+h} of j's merging onions at the end of the execution, i.e., $V^i_{L+h} = V^j_{L+h}$.*

Proof. Let ℓ be any epoch between the start of the merging phase and the partway point, and let $\nu_\ell = \frac{V_\ell}{U_\ell}$ be the ratio between the actual number V_ℓ of party i's onions at the ℓ-th diagnostic round and its upper bound, $U_\ell = \frac{\chi}{2^{\ell-L}} = \mathsf{polylog}(\lambda)$.

Fix $0 \leq \mathbb{E}[\zeta] \leq \frac{1}{2}$, and let $0 \leq \alpha \leq \mathbb{E}[\zeta]$ be any fraction between zero and $\mathbb{E}[\zeta]$. We will first analyze what happens when the adversary $\mathcal{A}$ drops α fraction of the remaining singletons during the $(\ell+1)$-st epoch (between the ℓ-th diagnostic and the $(\ell+1)$-st diagnostic) and another $\frac{\mathbb{E}[\zeta]-\alpha}{1-\alpha}$ fraction of the remaining singletons during the $(\ell+2)$-nd epoch.

At the ℓ-th diagnostic round, there are an expected (approx.) $\nu_\ell^2 U_\ell$ paired onions and an expected (approx.) $\nu_\ell(1-\nu_\ell)U_\ell$ singletons (Lemma 5). If the adversary $\mathcal{A}$ drops α fraction of the singletons, then for any small constant $\delta \geq \frac{V_R}{V_R - 1} - 1$, w.o.p.,

$$\begin{aligned}
\nu_{\ell+1} &\geq \frac{U_\ell}{U_{\ell+1}} \left((1-\delta)\frac{\nu_\ell^2}{2} + (1-\delta)(1-\alpha)\nu_\ell(1-\nu_\ell) \right) \\
&= 2(1-\delta)\left(\frac{\nu_\ell^2}{2} + (1-\alpha)\nu_\ell(1-\nu_\ell) \right) \\
&= (1-\delta)\nu_\ell^2 + 2(1-\delta)\nu_\ell - 2(1-\delta)\nu_\ell^2 - 2(1-\delta)\alpha\nu_\ell + 2(1-\delta)\alpha\nu_\ell^2 \qquad (3.9) \\
&\stackrel{\text{def}}{=} \xi_{\ell+1}.
\end{aligned}$$

At the $(\ell+1)$-st diagnostic round, there are an expected (approx.) $\nu_{\ell+1}^2 U_{\ell+1}$ paired onions and an expected (approx.) $\nu_{\ell+1}(1-\nu_{\ell+1})U_{\ell+1}$ singletons (Lemma 5). So if the

adversary $\mathcal{A}$ drops $\beta \stackrel{\text{def}}{=} \frac{\mathbb{E}[\zeta]-\alpha}{1-\alpha}$ fraction of the singletons, then w.o.p.,

$$\begin{aligned}
\nu_{\ell+2} &\geq \frac{U_{\ell+1}}{U_{\ell+2}} \left((1-\delta)\frac{\xi_{\ell+1}^2}{2} + (1-\delta)\,(1-\beta)\,\xi_{\ell+1}(1-\xi_{\ell+1}) \right) \\
&= 2(1-\delta) \left(\frac{\xi_{\ell+1}^2}{2} + (1-\beta)\,\xi_{\ell+1}(1-\xi_{\ell+1}) \right) \\
&= 2(1-\delta)\,(1-\beta)\,\xi_{\ell+1} + 2(1-\delta)\left(\beta - \frac{1}{2}\right)\xi_{\ell+1}^2 \\
&\stackrel{\text{def}}{=} \xi_{\ell+2}.
\end{aligned}$$

Taking a derivative of $\xi_{\ell+2}$ with respect to α, we get

$$\begin{aligned}
\frac{\partial \xi_{\ell+2}}{\partial \alpha} &= \left(\frac{\partial \xi_{\ell+2}}{\partial \xi_{\ell+1}}\right)\left(\frac{\partial \xi_{\ell+1}}{\partial \alpha}\right) \\
&= \left(2(1-\delta)\,(1-\beta+(2\beta-1)\xi_{\ell+1})\right)\left(2(1-\delta)\left(\nu_\ell^2-\nu_\ell\right)\right) \\
&= 4(1-\delta)^2\left(\nu_\ell^2-\nu_\ell\right)(1-\beta+(2\beta-1)\xi_{\ell+1}) \\
&\leq 0, \qquad (3.10)
\end{aligned}$$

since $\frac{\partial \xi_{\ell+1}}{\partial \alpha} = 2(1-\delta)\,(1-\beta+(2\beta-1)\xi_{\ell+1})$ from (3.9). This last inequality follows because $(1-\delta)^2 \geq 0$, $\left(\nu_\ell^2-\nu_\ell\right) \leq 0$ since $\nu_\ell \leq 1$ and $\xi_{\ell+1} \leq \frac{1-\beta}{1-2\beta}$ since $\beta = \frac{\mathbb{E}[\zeta]-\alpha}{1-\alpha} \leq \frac{1}{2}$. We now prove the lemma.

Case 1: The honest parties abort by the beginning of the merging phase. In this case, w.o.p., no honest onion will remain at the partway point. Thus, $V_R^i = 0$ for every honest party i.

Case 2: There is an unaborted honest party at the start of the merging phase. For every $L < \ell \leq R$, let α_ℓ be the fraction of singletons that the adversary drops in the ℓ-th epoch. Let $\mathbb{E}[\zeta_0] = 0$, and for every $L \leq \ell \leq R$, let $\mathbb{E}[\zeta_\ell] = \sum_{\tau=1}^{\ell}(1-\mathbb{E}[\zeta_{\tau-1}])\alpha_\tau$.

If $\mathbb{E}[\zeta_R] \geq \frac{1}{2}$: From Lemma 4, w.o.p., every honest party aborts the protocol run by the partway point. In this case, $V_{L+h}^i = 0$ for every honest party i.

If $\mathbb{E}[\zeta_R] < \frac{1}{2}$: From (3.10), the best that the adversary can do is to drop as many singletons (at most half) in the $(L+1)$-st epoch (all upfront) without causing the honest parties to abort the run. In this case, the adversary cannot afford to drop any more singletons past the $(L+1)$-st epoch.

From Lemma 3, if there is an unaborted honest party at the beginning of the merging phase, then w.o.p., a constant fraction of every honest party's merging onions remain at the beginning of the merging phase. Thus, w.o.p., $V_R^i = U_R$ for every honest party i. This last follows from a known concentration bound [HS05] for the hypergeometric distribution. $\square$

We now prove Theorem 9.

of Theorem 9. From Lemma 6, if during the merging phase, the adversary drops only singletons, then w.o.p., every recipient receives one merging onion in the end. Thus, to conclude the proof of Theorem 9, it suffices to show that the protocol equalizes when the adversary also drops mergeable pairs.

Fix an epoch $L < \ell \leq R$. Let $U \stackrel{\text{def}}{=} \frac{1}{2^{\ell-L}}$. For all honest i, let $V^i \stackrel{\text{def}}{=} \nu^i U$ be the number of i's merging onions remaining at the first round of the ℓ-th epoch.

Each mergeable pair that goes to an honest party in the first round of the epoch is merged into a single onion by round 2. Since the quantities of (indistinguishable) mergeable pairs and singletons at every party are close to their respective expected values (Chernoff bounds), the adversary can only drop these merged onions (necessarily after round 1) in proportion to singletons. The adversary can disproportionally drop more mergeable pairs only by dropping those that arrive at a corrupted party in the first round.

In the first round of the epoch, at most one-half of the mergeable pairs go to a corrupted party (Chernoff bounds). If the adversary drops all of these pairs, then the expected number $\mathbb{V}^j$ of j's mergeable onions at round 2 is given by $\mathbb{V}^j = V^j - \frac{\nu^j}{2} V^j = \nu^j U - \frac{(\nu^j)^2}{2} U$, and the expected number $\mathbb{V}^k$ of k's mergeable onions at round 2 is given by $\mathbb{V}^k = \nu^k U - \frac{(\nu^k)^2}{2} U$.

W.l.o.g., assume that $V^j \geq V^k$. It follows that $V^j - V^k \geq \mathbb{V}^j - \mathbb{V}^k$, and $\mathbb{V}^j - \mathbb{V}^k \geq 0$ since $U > 0$ by construction, and $\nu^k \leq \nu^j \leq 1$. Since the actual quantities are close to the expected values (Chernoff bound and Lemma 5), this shows that the protocol equalizes faster when the adversary also drops mergeable pairs compared to the scenario in which the adversary drops only singletons. □

3.6.4 Supplementary proofs

Proof of Lemma 4

Recall that κ is the upper bound on the corruption rate, and an *indistinguishable* onion is either an honest merging onion or an honest checkpoint onion with a checkpoint for verification by an honest party. A *distinguishable* onion is one that is not indistinguishable.

We first prove:

Lemma 7. *In $\Pi_{\bowtie}$: For any epoch $L < \ell \leq R$, no matter how the adversary drops singletons during an epoch, essentially, the adversary drops a random sample of the remaining singletons at the start of the epoch.*

Proof. We say that the $\Pi_{\bowtie}$ "*shuffles indistinguishable onions at round r_A by round r_B*" if for any two indistinguishable onions at r_B, given the adversarial view from r_A to r_B and the location at r_A of one of the onions chosen at random (plus the description of $\mathcal{A}$ and the input), no distinguisher can guess which onion was chosen with nonnegligible advantage.

The proof of Lemma 7 follows from combining Facts 1 and 2, below.

Fact 1: $\Pi_{\bowtie}$ shuffles indistinguishable onions at the first round by the start of the merging phase. Let a subnet in the butterfly network G be *good* if it consists of two honest parties and *so-so* if it consists of one honest party and one corrupted party. (In an unaborted execution) for every stage τ of G, every indistinguishable onion passes through a good or so-so subnet at stage τ a polylogarithmic number of times (Chernoff bound); thus, every bit of the onion's location at the start of the merging phase is independent of the other bits and equally likely to be a zero or a one.

Fact 2: For every epoch $L < \ell \leq R$, $\Pi_{\bowtie}$ shuffles the indistinguishable onions at the start of the epoch by the end of the epoch. By construction, the number of rounds and the expected number of honest onions per round and location are at least polylogarithmic in the security parameter; so shuffling follows from an earlier result from Ando, Lysyanskaya and Upfal [ALU18]. □

If the adversary drops ζ_ℓ fraction of the indistinguishable checkpoint onions by the ℓ-th diagnostic round, then every party would observe, on average, $(1-\kappa)\zeta_\ell \frac{\chi}{L+h}$ missing checkpoints at the ℓ-th diagnostic.

We now prove that, with overwhelming probability, the actual number of missing checkpoint onions is close to this expected quantity.

Lemma 8 (Two-color onions lemma)**.** *In $\Pi_{\bowtie}$: Suppose that the adversary drops at least a constant $0 \leq \zeta_\ell \leq 1$ fraction of all indistinguishable checkpoint onions before the ℓ-th diagnostic round (i.e., round ℓd).*

If F is a truly random function, then for all $0 < \delta \leq 1$, with overwhelming probability, each party k will notice at least between $(1-\delta)(1-\kappa)\zeta_\ell \frac{\chi}{L+h}$ and $(1+\delta)(1-\kappa)\zeta_\ell \frac{\chi}{L+h}$ missing checkpoints at the ℓ-th diagnostic round.

Proof. We recast this problem as a two-colored-balls problem. The different categories of balls correspond to different categories of onions (explained below).

Fix a party k and a diagnostic round ℓd.

The green balls/onions, $\mathcal{Z}$, are all the indistinguishable checkpoint onions for verification by party k at the ℓ-th diagnostic; let $Z = |\mathcal{Z}|$.

Let $\mathcal{Y} \supseteq \mathcal{Z}$ be all the indistinguishable checkpoint onions, including those in $\mathcal{Z}$; and let $Y = |\mathcal{Y}|$. The white onions/balls are the onions in $\mathcal{Y} \setminus \mathcal{X}$; these are the indistinguishable checkpoint onions *not* for verification by k at the ℓ-th diagnostic.

Since the onions in $\mathcal{Y}$ are indistinguishable, if the adversary drops ζ fraction of them, the adversary *eliminates* (or drops) a random sample $\mathcal{E} \subseteq \mathcal{Y}$ of size ζY.

Using a known concentration bound [HS05] for the hypergeometric distribution, when the expected number $\mathbb{E}[W] = \zeta Z$ of green balls in $\mathcal{E}$ is at least polylogarithmic in the security parameter, with overwhelming probability, the actual number W of green balls in $\mathcal{E}$ is close to $\mathbb{E}[W]$, i.e., $W = (1 \pm \delta')\,\mathbb{E}[W]$.

Let X be the number of distinguishable checkpoint onions. If

Claim 1. *X, Y and Z are close to their respective expected values, i.e., for any $0 < \delta' \leq 1$, with overwhelming probability, $X = (1 \pm \delta')\,\mathbb{E}[X]$, $Y = (1 \pm \delta')\,\mathbb{E}[Y]$, and $Z = (1 \pm \delta')\,\mathbb{E}[Z]$;*

then with overwhelming probability, at least $(1-\delta)(1-\kappa)\zeta_\ell \frac{\chi}{L+h}$ checkpoints onions will be missing for party k at the ℓ-th diagnostic.

To complete the proof, we now prove the claim above:

Let $\mathcal{Y}_1$ be the set of all (for all i's) indistinguishable checkpoint onions formed by party i for verification by party i, excluding the (possible) onion formed by party k for verification by party k at the ℓ-th diagnostic. Let $Y_1' = |\mathcal{Y}_1|$.

Let $\mathcal{Y}_2$ be the set of all (for all i's and all j's) indistinguishable checkpoint onions formed by party i for verification by party $j > i$, excluding any onion for verification by party k at the ℓ-th diagnostic as well as any onion *formed by* party k for verification at the ℓ-th diagnostic. Let $Y_2' = |\mathcal{Y}_2|$.

For every triple (i, j, τ) consisting of the index τ of a diagnostic round τd and honest parties i and j, let $Y_{i\to j}^{\tau}$ be one if party i forms a checkpoint onion to be verified by party j at the τ-th diagnostic (and zero, otherwise).

Since $Y_{i\to j}^{\tau} = 1 \iff Y_{j\to i}^{\tau} = 1$ (i.e., party i creates an onion to be verified by party j at the τ-th epoch iff party j creates a symmetric onion to be verified by party i at the τ-th epoch), it follows that the total (over all i's, all j's, and all τ's) number of checkpoint onions formed by party i for party $j \neq i$ is $2\left(Y_2' + \sum_{i\neq k} Y_{i\to k}^{\ell}\right)$.

Let $Y_1 = Y_1' + Y_{k\to k}^{\ell}$, and let $Y_2 = Y_2' + \sum_{i\neq k} Y_{i\to k}^{\ell}$. The total number Y of indistinguishable checkpoint onions is given by

$$Y = (Y_1' + Y_{k\to k}^{\ell}) + 2\left(Y_2' + \sum_{i\neq k} Y_{i\to k}^{\ell}\right) = Y_1 + 2Y_2. \tag{3.11}$$

If F is a truly random function, the onions in $\mathcal{Z} \cup \mathcal{Y}_1 \cup \mathcal{Y}_2$, are i.i.d. Bernoulli random variables, each having probability $q = \frac{2\chi}{N(L+h)}$ of success. It follows that

$$\begin{aligned} Z &\sim \mathsf{Binomial}\left((1-\kappa)N, q\right), \\ Y_1 &\sim \mathsf{Binomial}\left((1-\kappa)N(L+h), q\right), \\ Y_2 &\sim \mathsf{Binomial}\left(\binom{(1-\kappa)N}{2}(L+h), q\right). \end{aligned}$$

Using Chernoff bound for Poisson trials, for any $0 < \delta'' \leq 1$:

$$\Pr\left[|Z - \mathbb{E}[Z]| > \delta'' \mathbb{E}[Z]\right] \leq 2\exp\left(-\mathsf{polylog}(\lambda)\right) = \mathsf{negl}(\lambda) \tag{3.12}$$

$$\Pr\left[|Y_1 - \mathbb{E}[Y_1]| > \delta'' \mathbb{E}[Y_1]\right] \leq 2\exp\left(-\mathsf{polylog}(\lambda)\right) = \mathsf{negl}(\lambda) \tag{3.13}$$

$$\Pr\left[|Y_2 - \mathbb{E}[Y_2]| > \delta'' \mathbb{E}[Y_1]\right] \leq 2\exp\left(-\mathsf{polylog}(\lambda)\right) = \mathsf{negl}(\lambda). \tag{3.14}$$

Thus, with overwhelming probability, (i) $Z = (1 \pm \delta'')\,\mathbb{E}[Z] = (1-\delta'')(1-\kappa)\frac{2\chi}{L+h}$, (ii) $Y_1 = (1 \pm \delta'')\,\mathbb{E}[Y_1]$ and (iii) $Y_2 = (1 \pm \delta'')\,\mathbb{E}[Y_2]$.

Facts (ii) and (iii) imply

$$\begin{aligned} Y &= (1 \pm \delta'')(\mathbb{E}[Y_1] + 2\,\mathbb{E}[Y_2]) && (3.15) \\ &= (1 \pm \delta'')\left((1-\kappa)N(L+h) + 2\left(\frac{(1-\kappa)N((1-\kappa)N-1)}{2}(L+h)\right)\right)q && (3.16) \\ &= (1 \pm \delta'')(1-\kappa)^2 N^2 (L+h)\left(\frac{2\chi}{N(L+h)}\right) \\ &= (1 \pm \delta'') \cdot 2(1-\kappa)^2 \chi N, \end{aligned}$$

where (3.15) follows (3.11) and (3.12)-(3.14), and (3.16) holds because

$$\mathbb{E}[Y_1] = (1-\kappa)N(L+h)q$$

and

$$\mathbb{E}[Y_2] = \left(\frac{(1-\kappa)N((1-\kappa)N-1)}{2}(L+h)\right)q.$$

Following a similar argument as above, we have $X = (1+\delta')\,\mathbb{E}[X]$. This concludes are proof. □

Combining Lemmas 7 and 8 proves Lemma 4(ii). To prove Lemma 4(i), we require another lemma (below).

Recall that a "*singleton*" is an onion that does not belong in any mergeable pair; it is either a checkpoint onion, or a merging onion without a "mate" at the ℓ-th epoch.

Let $\mathcal{I}_\ell$ be the set of indistinguishable singletons (onion evolutions) at the ℓ-th epoch, and let Y be the number of indistinguishable checkpoint onions (onion evolutions) that are formed.

Suppose that, for every epoch ℓ, the adversary drops α_ℓ fraction of the onions in $\mathcal{I}_\ell$. Then, we expect that the adversary drops α_1 of the Y indistinguishable checkpoint onions during epoch 1, and another α_2 of the remaining $(1-\alpha_1)Y$ indistinguishable checkpoint onions during epoch 2, and so on. Following this logic, *by* the ℓ-th epoch, we expect that $\mathbb{E}[\zeta_\ell]$ fraction of the Y indistinguishable checkpoint onions have been dropped, where $\mathbb{E}[\zeta_\ell]$ is defined recursively as follows:

$$\mathbb{E}[\zeta_1] = 0 \tag{3.17}$$

$$\mathbb{E}[\zeta_\ell] = \sum_{\tau=1}^{\ell-1}(1-\mathbb{E}[\zeta_\tau])\alpha_\tau, \qquad \ell \geq 2. \tag{3.18}$$

This next lemma states that the actual fraction ζ_ℓ is close to the expected, $\mathbb{E}[\zeta_\ell]$.

Lemma 9. *In $\Pi_{\bowtie}$: For every epoch ℓ, let α_ℓ be the fraction of remaining indistinguishable singletons that the adversary drops during the ℓ-th epoch, and let $\mathbb{E}[\zeta_\ell]$ be as defined by* (3.17) *and* (3.18)*.*

The fraction ζ_ℓ of all indistinguishable checkpoint onions that the adversary drops by the ℓ-th epoch is close to $\mathbb{E}[\zeta_\ell]$, i.e., for all $0 < \delta \leq 1$, with overwhelming probability,

$$(1-\delta)\,\mathbb{E}[\zeta_\ell] \leq \zeta_\ell \leq (1+\delta)\,\mathbb{E}[\zeta_\ell]\,.$$

Proof. The proof is by induction.

Base case ($\ell = 2$) This follows from a known concentration bound [HS05] for the hypergeometric distribution.

Inductive step ($\ell > 2$) Assume that $(1-\delta)\,\mathbb{E}[\zeta_{\ell-1}] \leq \zeta_{\ell-1} \leq (1+\delta)\,\mathbb{E}[\zeta_{\ell-1}]$.

Let α'_ℓ be the fraction of the (remaining) indistinguishable checkpoint onions that the adversary drops during the ℓ-th epoch.

$$\begin{aligned}
\zeta_\ell &= \zeta_{\ell-1} + (1-\zeta_{\ell-1})\alpha'_\ell \\
&\geq (1-\delta)\,\mathbb{E}[\zeta_{\ell-1}] + (1-(1+\delta)\,\mathbb{E}[\zeta_{\ell-1}])\alpha'_\ell && (3.19)\\
&\geq (1-\delta)\,\mathbb{E}[\zeta_{\ell-1}] + (1-\delta)(1-(1+\delta)\,\mathbb{E}[\zeta_{\ell-1}])\alpha_\ell && (3.20)\\
&\geq (1-\delta)\,\mathbb{E}[\zeta_{\ell-1}] + (1-\delta)\alpha_\ell - (1-\delta)\,\mathbb{E}[\zeta_{\ell-1}]\,\alpha_\ell && (3.21)\\
&= (1-\delta)\,\mathbb{E}[\zeta_\ell]\,,
\end{aligned}$$

where (3.19) follows from the inductive hypothesis, (3.20) follows from a known concentration bound [HS05] for the hypergeometric distribution, and (3.21) follows because $1 - \delta^2 \geq 1 - \delta$.

We obtain the upper bound in a similar fashion. □

If $\zeta_\ell \geq \frac{1}{2}$, then with overwhelming probability, every honest party aborts the protocol before the $(\ell + 1)$-st epoch:

From Lemma 7, the adversary essentially drops a random sample of the remaining singletons. From Lemma 9, the actual fraction ζ_ℓ of indistinguishable checkpoint onions that have been dropped by the ℓ-th epoch is close to the expected faction, $\mathbb{E}[\zeta_\ell] \geq (1-\delta)\zeta$. From Lemma 8, each party will notice close to the expected number of missing checkpoints: $(1-\delta)\,\mathbb{E}[\zeta_\ell]\,\frac{\chi}{L+h}$ (for an arbitrarily small δ).

Proof of Lemma 5

Proof. Let $v = \frac{|\mathcal{V}|}{2}$. For every $i \in [u]$, let w_i be one if both onions that comprise the u-th pair in $\mathcal{U}$ are in $\mathcal{V}$, and zero otherwise.

$$\begin{aligned}\mathbb{E}[w_i] = \Pr[w_i = 1] = \frac{\binom{2u-2}{2v-2}}{\binom{2u}{2v}} &= \frac{(2u-2)!}{(2v-2)!(2u-2v)!} \cdot \frac{(2v)!(2u-2v)!}{(2u)!} \\ &= \frac{2v(2v-1)}{2u(2u-1)},\end{aligned}$$

since there are $\binom{2u-2}{2v-2}$ ways to choose $2v-2$ balls from $2u-2$ balls; and likewise, there are $\binom{2u}{2v}$ ways of choosing $2v$ balls from $2u$ balls.

Let w denote the number of pairs in $\mathcal{V}$ From the linearity of expectation,

$$\mathbb{E}[w] = \sum_{i=1}^{u} \mathbb{E}[w_i] = u \cdot \frac{2v(2v-1)}{2u(2u-1)} = \frac{2v-1}{2u-1} \cdot v.$$

Recall that $\nu = \frac{v}{u} = \frac{|\mathcal{V}|}{|\mathcal{U}|}$. It follows that

$$\mathbb{E}[W] = \left(\frac{2v-1}{2u-1}\right)|\mathcal{V}| = \left(\frac{2v-1}{2v-\nu}\right)\left(\frac{\nu(2u-1)}{2u-1}\right)|\mathcal{V}| = \left(\frac{2v-1}{2v-\nu}\right)\nu|\mathcal{V}|.$$

For each $i \in [2v]$, let Y_i be the i-th chosen ball in $\mathcal{V}$, and let

$$Z_i = \mathbb{E}[W|Y_1, Y_2, \ldots, Y_i]\,.$$

Then, $Z_0, Z_1, \ldots, Z_{2v}$ is a Doob martingale by construction satisfying the Lipschitz condition with bound 1. Thus, from the Azuma-Hoeffding inequality, for any $0 < \delta \leq 1$,

$$\begin{aligned}\Pr[|W - \mathbb{E}[W]| \geq \delta\, \mathbb{E}[W]] \leq 2\exp\left(-\frac{\delta^2\, \mathbb{E}[W]^2}{\sum_{i=1}^{2v} 1}\right) &= 2\exp\left(-\frac{\delta^2(2v-1)^2}{(2u-1)^2}\cdot 2v\right) \\ &= 2\exp\left(-\Theta(1)\cdot 2v\right) \\ &= \mathsf{negl}(\lambda).\end{aligned}$$

This completes our proof. □

Proof that $\Pi_\triangle$ is anonymous

We will show that if the adversary does not drop too many honest onions before the first diagnostic round, then for each honest party, a constant fraction of her merging onions will survive (not be dropped in) the first epoch.

Our proof essentially boils down to proving that the following undesirable events rarely happens:

(i) For any honest party, the onions formed by the party do not travel together.

(ii) The first diagnostic fails to detect that the adversary dropped too many honest onions.

Below, we show that events (i) and (ii) can occur with only negligible probability.

Recall that an *indistinguishable* onion is either an honest merging onion or an honest checkpoint onion with a checkpoint for verification by an honest party.

Lemma 10. *Let λ be the security parameter, and let $N' = (1-\kappa)N \leq \mathsf{poly}(\lambda)$ be the number of honest parties (or locations). Let $\mathcal{O}$ be the set of indistinguishable onions at any round r of the protocol execution. If $|\mathcal{O}| = \Omega\left(\log^2 \lambda\right)$ and $|\mathcal{O}| \leq (N'+2)/2$, then, with overwhelming probability in λ, at least $|\mathcal{O}|/\log\lambda$ parties receive at least one onion from $\mathcal{O}$.*

Proof. Let $X = |\mathcal{O}|$ be the size of $\mathcal{O}$.

We recast this problem as a balls-and-bins problems, where the onions in $\mathcal{O}$ are the balls, and the parties (or locations) are the bins. To prove the lemma, we show that when $X = |\mathcal{O}|$ balls are thrown independently and uniformly at random into the N' bins, with overwhelming probability in λ, the number of non-empty bins is at least $\frac{X}{\log\lambda}$.

Let $L_1, \ldots, L_{N'}$ be the N' bins. For each $i \in [N']$, let X_i be the indicator random variable that is one if the i-th bin L_i is *empty* (and zero, otherwise). The probability that L_i remains empty is given as $\Pr[X_i = 1] = \mathbb{E}[X_i] = \left(1 - \frac{1}{N'}\right)^X$. The total number

$X = \sum_{i=1}^{N'} X_i$ of empty bins is the summation of all the X_i's. By the linearity of expectation, $\mathbb{E}[X] = \sum_{i=1}^{N'} \mathbb{E}[X_i] = N' \left(1 - \frac{1}{N'}\right)^X$.

Let W be the total number of *non-empty* bins; i.e., $W = N' - X$. Again by linearity of expectation,

$$\begin{aligned}
\mathbb{E}[W] &= N' - \mathbb{E}[X] \\
&= N' - N' \left(1 - \frac{1}{N'}\right)^X \\
&> N' - N' \left(1 - \frac{X}{N'} + \frac{X(X-1)}{N'^2}\right) && (3.22) \\
&= X \left(1 - \frac{X-1}{N'}\right) \\
&\geq \frac{X}{2}, && (3.23)
\end{aligned}$$

where (3.22) holds since $\left(1 - \frac{1}{N'}\right)^X$ is strictly less than the first three terms of its Laurent series, and (3.23) holds since $X \leq \frac{N'+2}{2}$ by the hypothesis.

For every $i \in [X]$, let Y_i be the location of the i-th ball, and let

$$Z_i = \mathbb{E}[X | Y_1, \ldots, Y_i] .$$

The sequence $Z_1, \ldots, Z_X$ is a Doob martingale by construction, satisfying the Lipschitz condition with constant bound one, i.e., $|Z_{i-1} - Z_i| \leq 1$. Thus, we may apply the Azuma-Hoeffding inequality as follows: For $\delta \geq \frac{\log \lambda - 2}{\log \lambda}$,

$$\begin{aligned}
\Pr[X - \mathbb{E}[X] \geq \delta \, \mathbb{E}[W]] &\leq \mathsf{exp}\left(-\frac{\delta^2 \, \mathbb{E}[W]^2}{2 \sum_{i=1}^{X} 1}\right) \\
&\leq \mathsf{exp}\left(-\frac{\delta^2 (X/2)^2}{2X}\right) && (3.24) \\
&= \mathsf{exp}\left(-\frac{\delta^2 X}{8}\right) \\
&\leq \mathsf{exp}\left(-\frac{(\log \lambda - 2)^2 \alpha \log^2 \lambda}{8 \log^2 \lambda}\right) && (3.25) \\
&= \mathsf{exp}\left(-\frac{\alpha (\log \lambda - 2)^2}{8}\right) \\
&= \mathsf{negl}(\lambda) ,
\end{aligned}$$

where (3.24) follows directly from (3.23), and (3.25) holds since $\delta \geq \frac{\log \lambda - 2}{\log \lambda}$ and $X \geq \alpha \log^2 \lambda$ from the hypothesis.

In other words, with overwhelming probability in λ,

$$X - \mathbb{E}[X] \leq \delta \, \mathbb{E}[W] \, .$$

Thus, it follows that, with overwhelming probability in λ,

$$\begin{aligned} W &\geq (1-\delta) \, \mathbb{E}[W] \\ &= \left(1 - \frac{\log \lambda - 2}{\log \lambda}\right) \mathbb{E}[W] \\ &= \frac{2}{\log \lambda} \, \mathbb{E}[W] \\ &\geq \frac{X}{\log \lambda}, \end{aligned} \tag{3.26}$$

where (3.26) follows directly from (3.23). □

Recall that χ is the number of merging onions formed by each honest party, and $h = \log \chi + 1$ is the number of diagnostic rounds (or the number of epochs)[4] in a full unaborted execution of $\Pi_{\triangle}$.

Lemma 11 (Italian onions)**.** *In $\Pi_{\triangle}$: Suppose that the adversary drops ζ fraction of all indistinguishable onions before the first diagnostic such that $\zeta^{\frac{\chi}{h}} = \mathsf{polylog}(\lambda)$.*

If F is a truly random function, then for all $0 < \delta \leq 1$, with overwhelming probability in λ, each honest party k will notice at least $(1-\delta)(1-\kappa)\zeta^{\frac{\chi}{h}}$ missing checkpoints at the first diagnostic.

Proof. We recast this problem as a three-colored-balls problem with green balls, white balls and red balls. The different categories of balls correspond to different categories of onions (explained below).

Fix an honest party k.

Let $\mathcal{Z}$ be the set of all indistinguishable checkpoint onions for verification by party k at the first diagnostic. (These correspond to the green onions/balls. All other indistinguishable checkpoint onions belong to the set $\mathcal{Y}$ and are the white onions/balls.) Let $Z = |\mathcal{Z}|$.

If F is a truly random function,

$$Z \sim \mathsf{Binomial}\left((1-\kappa)N, q\right).$$

Using Chernoff bound for Poisson trials, for any $0 < \delta' \leq 1$:

$$\Pr\left[|Z - \mathbb{E}[Z]\,| > \delta' \, \mathbb{E}[Z]\right] \leq 2 \exp\left(-\mathsf{polylog}(\lambda)\right) = \mathsf{negl}(\lambda) \, . \tag{3.27}$$

[4] e.g., when $\chi = 2$, there are two epochs, one corresponding to the leaf node and another to the root node of $G_{\triangle}$.

Thus, with overwhelming probability, Z falls between $(1-\delta')\,\mathbb{E}[Z] = (1-\delta')(1-\kappa)\frac{\chi}{h}$ and $(1+\delta')\,\mathbb{E}[Z] = (1+\delta')(1-\kappa)\frac{\chi}{h}$.

Let $\mathcal{X}$ be the set of all honest merging onions. (These correspond to the red onions/balls.)

Let $\mathcal{I}$ be the set of all indistinguishable onions, i.e., the set of all green, white and red onions/balls.

Since the adversary cannot distinguish between any two onions in $\mathcal{I}$, the cumulative set $\mathcal{E} \subseteq \mathcal{I}$ of indistinguishable onions that are *eliminated* (or dropped) by the adversary by the first diagnostic is a random subset of the set $\mathcal{I}$ of all honest onions.

Let $\zeta = \frac{|\mathcal{E}|}{|\mathcal{I}|}$ be the fraction of onions in $\mathcal{I}$ dropped by the adversary by the first diagnostic.

Using a known concentration bound for the hypergeometric distribution [HS05], when the expected number ζZ of green balls in $\mathcal{E}$ is at least polylogarithmic in λ, i.e., $\zeta Z = \mathsf{polylog}(\lambda)$, the actual number W of green balls in $\mathcal{E}$ is close to the expected value ζZ, i.e., for any $0 < \delta' \leq 1$, $(1-\delta')\zeta Z \leq W \leq (1+\delta')\zeta Z$. Combining this with (3.27) above, with overwhelming probability, the number of green balls in the random sample $\mathcal{E}$ falls between $(1-\delta')^2(1-\kappa)\zeta\frac{\chi}{h}$ and $(1-\delta')^2(1-\kappa)\zeta\frac{\chi}{h}$.

By choosing an appropriate δ such that $1-\delta \leq (1-\delta')^2$ and $1+\delta \geq (1+\delta')^2$, we obtain our desired bound. □

We will now prove the main theorem.

Theorem 10. *In $\Pi_{\triangle}$: Let λ denote the security parameter, and let ϵ, δ be small positive constants.*

Let every honest party form $\chi = 2^{\lceil \log A(\log A+1)\rceil}$ merging onions (and an expected χ checkpoint onions), where $A = \max((N\log^{1+\epsilon}\lambda)^{1/2}, \log^{2+\epsilon}\lambda)$. Let $T = 2(1-\delta)(1-\kappa)^3\kappa\log^{1+\epsilon}\lambda$ be the threshold for the first diagnostic; if at least T checkpoints are missing for an honest party k, then k aborts.

For every party i, let $V^{\mathcal{A},i}$ denote the number of i's merging onions at the first diagnostic round in an interaction with the adversary $\mathcal{A}$ who drops up to $\kappa < 1$ fraction of the parties. If there exists an unaborted honest party at the second epoch, then for any constant $0 < \delta \leq 1$, w.o.p.,

$$V^{\mathcal{A},i} \geq (1-\delta)(1-\kappa)^2\chi, \qquad \forall \mathcal{A}, \forall \text{ honest } i.$$

Proof. Fix an input σ and an honest party i.

Let N be the number of parties, and let $\mathcal{A}$ be the adversary who corrupts a random $\lfloor \kappa N \rfloor$ parties and at rounds 1 and 2, drops every droppable (delivered to a corrupted node)

onion that could have been formed by party i (from the adversary's perspective).

Let $X(r)$ be the number of indistinguishable onions at round r that that could have been formed by party i. In the first round, i transmits $X(1)$ onions. For any arbitrarily small $0 < \delta' \leq 1$, with overwhelming probability, $X(1) \geq 2(1-\delta')(1-\kappa)\chi$ because, with overwhelming probability, i forms at least $(1-\delta')(1-\kappa)\chi$ checkpoint onions with a checkpoint for an honest party (Chernoff bound) and χ merging onions.

Let the *span* at round r, denoted $S(r)$, be the number of honest parties that receive an indistinguishable onion that could have been formed by party i at round r. From Chernoff bound, at least $(1-\delta')(1-\kappa)$ of these $X(1)$ onions go to honest parties. So, from Lemma 10,

$$S(1) \geq 2(1-\delta')^2(1-\kappa)^2\frac{\chi}{\log\lambda}. \tag{3.28}$$

Each of these $S(1)$ parties receives at least $2(1-\delta')(1-\kappa)\chi$ indistinguishable onions at round 1 (Chernoff bound). Combining this with (3.28), there are at least $4(1-\delta')^3(1-\kappa)^3\frac{\chi^2}{\log\lambda}$ indistinguishable onions that could have originated from party i at round 2; that is,

$$X(2) \geq 4(1-\delta')^3(1-\kappa)^3\frac{\chi^2}{\log\lambda}.$$

At least $(1-\delta')\kappa$ of these onions are routed to corrupted parties at round 2 (Chernoff bound); that is, the number of indistinguishable onions from $X(2)$ that go to corrupted parties is at least

$$\begin{aligned}
4(1-\delta')^4(1-\kappa)^3\kappa\frac{\chi^2}{\log\lambda} &\geq 4(1-\delta')^4(1-\kappa)^3\kappa\frac{A^2}{\log\lambda}(\log A+1)^2 \\
&\geq 4(1-\delta')^4(1-\kappa)^3\kappa(N\log^{1+\epsilon}\lambda)(\log A+1)^2 \\
&\geq 4(1-\delta')^4(1-\kappa)^3\kappa(N\log^{1+\epsilon}\lambda)h,
\end{aligned}$$

where $A \geq \sqrt{N\log^{1+\epsilon}\lambda}$, and $h = \log\chi + 1$. If all of them are dropped, the fraction ζ of the indistinguishable onions that $\mathcal{A}$ drops is at least

$$\begin{aligned}
\zeta &\geq \frac{4(1-\delta')^4(1-\kappa)^3\kappa(N\log^{1+\epsilon}\lambda)h}{2(1+\delta')\chi N} \\
&= \left(\frac{4(1-\delta')^4(1-\kappa)^3\kappa}{2(1+\delta')}\right)\log^{1+\epsilon}\lambda\cdot\frac{h}{\chi},
\end{aligned}$$

because there are fewer than $2(1+\delta')\chi$ indistinguishable onions in total (Chernoff bound).

Let $1-\delta = \frac{(1-\delta')^5}{1+\delta'}$. From Lemma 11, each honest party k will notice at least

$$\begin{aligned}
(1-\delta')\zeta\frac{\chi}{h} &\geq \left(\frac{2(1-\delta')^5(1-\kappa)^3\kappa}{1+\delta'}\right)\log^{1+\epsilon}\lambda\cdot\frac{h}{\chi}\cdot\frac{\chi}{h} \\
&= 2(1-\delta)(1-\kappa)^3\kappa\log^{1+\epsilon}\lambda
\end{aligned}$$

missing checkpoints and will, therefore, abort the protocol.

Any adversary that drops at least as many onions as $\mathcal{A}$ will cause the honest parties to abort the protocol.

An adversary $\mathcal{A}'$ that drops at most as many onions as $\mathcal{A}$ can only do worse than $\mathcal{A}$; if $\mathcal{A}'$ deviates from $\mathcal{A}$ either by dropping fewer onions or waiting to drop onions, then

$$V^{\mathcal{A}',i} \geq V^{\mathcal{A},i}. \tag{3.29}$$

For any $0 < \delta \leq 1$, at least $(1-\delta)(1-\kappa)^2$ of party i's merging onions are randomly routed through only honest parties in rounds 1 and 2 (Chernoff bound); it follows that

$$V^{\mathcal{A},i} \geq (1-\delta)(1-\kappa)^2\chi. \tag{3.30}$$

Combining (3.29) and (3.30), we obtain our desired result. □

Since merging onions equalize (see §3.6), it follows that $\Pi_{\triangle}$ is anonymous in this setting.

3.7 Concluding remarks

We initiated a rigorous theoretical study of onion routing by providing new definitions and both lower and upper bounds.

Here, we point out a few extensions to our results: We proved that the required onion cost for an OR protocol to provide robustness and (computational) anonymity from the active adversary is polylogarithmic in the security parameter. Our proof for the lower bound can be used to prove the stronger result that polylogarithmic onion cost is required even when (1) the adversary observes the traffic on only $\Theta(1)$ fraction of the links and or when (2) the security definition is weakened to (computational) differential privacy. (3) Also, while we explicitly showed this to be the case for the simple I/O setting, the result holds more generally whenever any party can send a message to any other party.

We also proved the existence of a robust and anonymous OR protocol with polylogarithmic (in the security parameter) onion cost. (4) This result also extends beyond the simple I/O setting; our OR protocol is anonymous w.r.t. any input set where the size of each party's input is fixed.

There is a small gap between our lower and upper bounds. A natural direction for future work is to close this gap.

Chapter 4

Repliable OE Schemes

.

4.1 Contribution overview and related work

In this chapter, we present the first formal treatment of provably secure "repliable" onion encryption schemes. Our work resolves the problem of formalizing onion encryption for two-way channels.

4.1.1 Related work

In 1981, David Chaum introduced onion routing and mixes [Cha81]. As part of this work, Chaum provided a *reply option*, that is, a mechanism for the recipient of an onion to reply to the anonymous sender. However, Chaum's scheme, as well as that of the subsequent implementation, Cyberpunk anonymous remailer (a.k.a. Type I anonymous remailer) [Par96], are not cryptographically secure as defined by the later work of Camenisch and Lysyanskaya [CL05]. An *anonymous remailer* is a server that processes message packages that are essentially onions. There are many examples of anonymous remailers in practice, notably Cyberpunk (Type I), Mixmaster (Type II) [Cot95, MC00] and Mixminion (Type III) [DDM03], each defined by the algorithms they run.

In their 2005 paper [CL05], Camenisch and Lysyanskaya (CL) provided the first formal treatment of onion encryption; they presented the input/output (I/O) syntax of onion encryption schemes for one-way anonymous channels and an ideal functionality $\mathcal{F}_{\mathsf{onion}}$ of an onion encryption schemes in Canetti's universal composability (UC) framework [Can01]. Additionally, they gave a set of cryptographic definitions – onion-correctness, onion-integrity

and onion-security – that they claimed imply the realizability of the ideal functionality $\mathcal{F}_{\mathsf{onion}}$. Finally, they also provided a construction that they showed satisfied these cryptographic properties. However, their construction, as well as the subsequent implementation, Mixmaster anonymous remailer (a.k.a. Type II anonymous remailer) [Cot95, MC00], do not provide a reply option.

In a recent paper, Kuhn et al. showed that CL's set of cryptographic definitions was incomplete [KBS19]. Whereas CL's onion-security prevents the adversary from learning whether the honest party receiving the challenge onion is an intermediary of the onion or its recipient, it doesn't symmetrically protect the honest sender. As an example, Kuhn et al. pointed out this security flaw in the data packet format, Sphinx [DG09], which replaced the Mixminion anonymous remailer (a.k.a. Type III anonymous remailer) [DDM03]. Additionally, Kuhn et al. proposed a different set of cryptographic definitions that includes two new definitions – *tail-indistinguishability* (i.e., the adversary cannot tell whether the honest party is the sender or an intermediary of the challenge onion) and *layer-unlinkability* (i.e., onion layers are computationally unrelated to each other) – that together imply realizability of CL's ideal functionality $\mathcal{F}_{\mathsf{onion}}$. However, Kuhn et al. didn't provide any construction of their own and also left open the problem of formalizing onion encryption for two-way anonymous channels.

4.1.2 Our contributions

Our contributions are as follows:

1. In Section 4.2, we provide the I/O syntax for repliable onion encryption schemes and define what it means for a repliable onion encryption scheme to be correct.

2. At a high level, honestly formed onions in an onion routing protocol should mix at honest nodes. This property is what enables anonymity from the standard adversary who can observe the network traffic on all communication links. Ideally, onions should mix (i) even if the distances from their respective origins or the distances to their respective destinations differ, and (ii) regardless of whether they are forward or return onions. In Section 4.3, we give a detailed description of an ideal functionality, $\mathcal{F}_{\mathsf{ROES}}$, for repliable onion encryption schemes in the simplified UC model [CCL15]. We define the ideal functionality so that a scheme that realizes the ideal functionality necessarily satisfies properties (i) and (ii) above.

3. In Section 4.4, we present our construction, Shallot Encryption Scheme, and, in Section 4.5, prove that it UC-realizes $\mathcal{F}_{\mathsf{ROES}}$.

4. We prove that our construction is secure by first providing a cryptographic definition of security, *repliable-onion security* (Definition 16), that implies realizability of $\mathcal{F}_{\mathsf{ROES}}$ and proving that our construction is repliable onion-secure. Our scheme builds on a CCA2-secure cryptosystem with tags, a block cipher and a collision-resistant hash function.

Intuitively, an ideal functionality of repliable onion encryption should, firstly, inherit the security properties of onion encryption for one-way channels and, additionally, require the indistinguishability of return onions (from recipients to senders) from forward onions (from senders to recipients). Thus, to get an ideal functionality of onion encryption for two-way channels, a natural thing to try might be to build upon CL's ideal functionality $\mathcal{F}_{\mathsf{onion}}$ of onion encryption for one-way channels. Unfortunately, this approach does not work: The header (the encrypted routing information) of a return onion must be formed by the sender, whereas the return onion's content (the encrypted payload) must come from the possibly corrupt recipient. However, $\mathcal{F}_{\mathsf{onion}}$ does not handle "hybrid" onions with onion layers formed by both honest and corrupt parties. For this reason, it was necessary to redefine ideal onion encryption from the ground up, this time for two-way channels.

Our ideal functionality describes how onions with honestly formed headers should behave. The environment sends instructions (to form or process onions) to only honest parties, and, in the ideal setting, the ideal functionality keeps track of only onions with honestly formed headers. The environment directly handles all other onions (e.g., onions formed by corrupt parties or onions with mauled headers). With this new setting, we also avoid having to assume onion-integrity to prove that a cryptographic security definition implies UC-realizability of an ideal functionality. Even though Kuhn et al. claimed that onion-integrity is superfluous, the reduction that they provided should have used this property, and so they didn't explicitly show how to avoid requiring this property.

4.2 Repliable onion encryption: syntax and correctness

In this chapter, an onion O is a pair, consisting of the (encrypted) content C and the header H, i.e., $O = (H, C)$.

We assume that the upper bound N on the length of the forward or return path is one of the public parameters pp.

Here, we give the formal input/output (I/O) specification for a repliable onion encryption scheme. In contrast to the I/O specification for an (unrepliable) onion encryption scheme given by Camenisch and Lysyanskaya [CL05], a repliable onion encryption scheme contains an additional algorithm, $\mathsf{FormReply}$, for forming return onions. This algorithm allows the recipient of a message contained in a repliable onion to respond to the anonymous sender of the message without needing to know who the sender is.

The algorithm for forming onions, $\mathsf{FormOnion}$, also takes as one of its parameters, the label ℓ. This is so that when the sender receives a reply message m' along with the label ℓ, the sender can identify to which message m' is responding.

Definition 13 (Repliable onion encryption scheme I/O)**.** *The set* $\Sigma = (G, \mathsf{FormOnion}, \mathsf{ProcOnion}, \mathsf{FormReply})$ *of algorithms satisfies the I/O specification of a* repliable onion encryption scheme *for the label space* $\mathcal{L}(1^\lambda)$*, the message space* $\mathcal{M}(1^\lambda)$ *and a set* $\mathcal{P}$ *of router names if:*

- *G is a probabilistic polynomial-time (p.p.t.) key generation algorithm. On input the security parameter 1^λ (written in unary), the public parameters pp and the party name P, the algorithm G returns a key pair, i.e.,*

$$(\mathsf{pk}(P), \mathsf{sk}(P)) \leftarrow G(1^\lambda, \mathsf{pp}, P).$$

- $\mathsf{FormOnion}$ *is a p.p.t. algorithm for forming onions. On input*

  i. a label $\ell \in \mathcal{L}(1^\lambda)$ from the label space,

  ii. a message $m \in \mathcal{M}(1^\lambda)$ from the message space,

  iii. a forward path $P^{\rightarrow} = (P_1, \ldots, P_d)$ (d stands for destination),

  iv. the public keys $\mathsf{pk}(P^{\rightarrow})$ associated with the parties in $P^{\rightarrow}$,

  v. a return path $P^{\leftarrow} = (P_{d+1}, \ldots, P_s)$ (s stands for sender) and

  vi. the public keys $\mathsf{pk}(P^{\leftarrow})$ associated with the parties in $P^{\leftarrow}$,

  the algorithm $\mathsf{FormOnion}$ *returns a sequence $O^{\rightarrow} = (O_1, \ldots, O_d)$ of onions for the forward path, a sequence $H^{\leftarrow} = (H_{d+1}, \ldots, H_s)$ of headers for the return path and a key κ, i.e.,*

$$(O^{\rightarrow}, H^{\leftarrow}, \kappa) \leftarrow \mathsf{FormOnion}(\ell, m, P^{\rightarrow}, \mathsf{pk}(P^{\rightarrow}), P^{\leftarrow}, \mathsf{pk}(P^{\leftarrow}))$$

- $\mathsf{ProcOnion}$ *is a deterministic polynomial-time (d.p.t.) algorithm for processing onions. On input an onion O, a router name P and the secret key $\mathsf{sk}(P)$ belonging to P, the algorithm* $\mathsf{ProcOnion}$ *returns* $(\mathsf{role}, \mathsf{output})$*, i.e.,*

$$(\mathsf{role}, \mathsf{output}) \leftarrow \mathsf{ProcOnion}(O, P, \mathsf{sk}(P)).$$

When $\mathsf{role} = \mathsf{I}$ *(for "intermediary"),* output *is the pair (O', P') consisting of the peeled onion O' and the next destination P' of O'. When* $\mathsf{role} = \mathsf{R}$ *(for "recipient"),* output *is the message m for recipient P. When* $\mathsf{role} = \mathsf{S}$ *(for "sender"),* output *is the pair (ℓ, m) consisting of the label ℓ and the reply message m for sender P.*

- $\mathsf{FormReply}$ *is a d.p.t. algorithm for forming return onions. On input a reply message $m \in \mathcal{M}(1^\lambda)$, an onion O, a router name P and the secret key $\mathsf{sk}(P)$ belonging to P, the algorithm* $\mathsf{FormReply}$ *returns the peeled onion O' and the next destination P' of O', i.e.,*

$$(O', P') \leftarrow \mathsf{FormReply}(m, O, P, \mathsf{sk}(P)).$$

$\mathsf{FormReply}$ *outputs $(\perp, \perp)$ if O is "not repliable".*

4.2.1 Onion evolutions, forward paths, return paths and layerings

Now that we have defined the I/O specification for a repliable onion encryption scheme, we can define what it means for a repliable onion encryption scheme to be correct. Before we do this, we first define what onion *evolutions*, *paths* and *layerings* are; the analogous notions for the unrepliable onion encryption scheme were introduced by Camenisch and Lysyanskaya [CL05].

Let $\Sigma = (G, \mathsf{FormOnion}, \mathsf{ProcOnion}, \mathsf{FormReply})$ be a repliable onion encryption scheme for the label space $\mathcal{L}(1^\lambda)$, the message space $\mathcal{M}(1^\lambda)$ and the set $\mathcal{P}$ of router names.

Let there be honest parties with honestly formed keys.

Let $O_1 = (H_1, C_1)$ be an onion received by party P_1, not necessarily formed using $\mathsf{FormOnion}$.

We define a sequence of onion-location pairs recursively as follows: Let d be the first onion layer of (H_1, C_1) that when peeled, produces either "R" or "S" (if it exists, otherwise $d = \infty$). For all $i \in [d-1]$, let

$$(\mathsf{role}_{i+1}, ((H_{i+1}, C_{i+1}), P_{i+1})) = \mathsf{ProcOnion}((H_i, C_i), P_i, \mathsf{sk}(P_i)).$$

Let $s = d$ if peeling (H_d, C_d) produces "S". Otherwise, let $m \in \mathcal{M}(1^\lambda)$ be a reply message from the message space, and let

$$((H_{d+1}, C_{d+1}), P_{i+1}) = \mathsf{FormReply}(m, (H_d, C_d), P_d, \mathsf{sk}(P_d)).$$

Let $s = d$ if (H_d, C_d) is "not repliable", i.e., $((H_{d+1}, C_{d+1}), P_{i+1}) = (\perp, \perp)$. Otherwise, let s be the first onion layer of (H_{d+1}, C_{d+1}) that when peeled, produces either "R" or "S" (if it exists, otherwise $s = \infty$). For all $i \in \{d+1, \ldots, s-1\}$, let

$$(\mathsf{role}_{i+1}, ((H_{i+1}, C_{i+1}), P_{i+1})) = \mathsf{ProcOnion}((H_i, C_i), P_i, \mathsf{sk}(P_i)).$$

We call the sequence $\mathcal{E}(H_1, C_1, P_1, m) = ((H_1, C_1, P_1), \ldots, (H_s, C_s, P_s))$ of onion-location pairs the "*evolution of the onion* (H_1, C_1) *starting at party* P_1 *given* m *as the reply message*". The sequence $\mathcal{P}^{\rightarrow}(H_1, C_1, P_1, m) = (P_1, \ldots, P_d)$ is its *forward path*; the sequence $\mathcal{P}^{\leftarrow}(H_1, C_1, P_1, m) = (P_{d+1}, \ldots, P_s)$ is its *return path*; and the sequence $\mathcal{L}(H_1, C_1, P_1, m) = (H_1, C_1, \ldots, H_d, C_d, H_{d+1}, \ldots, H_s)$ is its *layering*.

We define correctness as follows:

Definition 14 (Correctness)**.** *Let* G*,* $\mathsf{FormOnion}$*,* $\mathsf{ProcOnion}$ *and* $\mathsf{FormReply}$ *form a repliable onion encryption scheme for the label space* $\mathcal{L}(1^\lambda)$*, the message space* $\mathcal{M}(1^\lambda)$ *and the set* $\mathcal{P}$ *of router names.*

Let N *be the upper bound on the path length (in* pp*). Let* $P = (P_1, \ldots, P_s)$*,* $|P| = s \leq 2N$ *be any list (not containing* $\perp$*) of router names in* $\mathcal{P}$*. Let* $d \in [s]$ *be any index in* $[s]$ *such that* $d \leq N$ *and* $s - d + 1 \leq N$*. Let* $\ell \in \mathcal{L}(1^\lambda)$ *be any label in* $\mathcal{L}(1^\lambda)$*. Let* $m, m' \in \mathcal{M}(1^\lambda)$ *be any two messages in* $\mathcal{M}(1^\lambda)$*.*

For every party P_i *in* P*, let* $(\mathsf{pk}(P_i), \mathsf{sk}(P_i)) \leftarrow G(1^\lambda, \mathsf{pp}, P_i)$*.*

Let $P^{\rightarrow} = (P_1, \ldots, P_d)$*, and let* $\mathsf{pk}(P^{\rightarrow})$ *be a shorthand for the public keys associated with the parties in* $P^{\rightarrow}$*. Let* $P^{\leftarrow} = (P_{d+1}, \ldots, P_s)$*, and let* $\mathsf{pk}(P^{\leftarrow})$ *be a shorthand for the public keys associated with the parties in* $P^{\leftarrow}$*.*

Let $((H_1, C_1), \ldots, (H_d, C_d), H_{d+1}, \ldots, H_s, \kappa)$ *be the output of* $\mathsf{FormOnion}$ *on input the label* ℓ*, the message* m*, the forward path* $P^{\rightarrow} = (P_1, \ldots, P_d)$*, the public keys* $\mathsf{pk}(P^{\rightarrow})$ *associated with the parties in* $P^{\rightarrow}$*, the return path* $P^{\leftarrow} = (P_{d+1}, \ldots, P_s)$ *and the public keys* $\mathsf{pk}(P^{\leftarrow})$ *associated with the parties in* $P^{\leftarrow}$*.*

The scheme Σ *is* correct *if with overwhelming probability in the security parameter* λ*,*

i. ***Correct forward path.***

- $\mathcal{P}^{\rightarrow}(H_1, C_1, P_1, m') = (P_1, \ldots, P_d)$.

- *For every $i \in [d]$ and content C such that $|C| = |C_i|$, $\mathcal{P}^{\rightarrow}(H_i, C, P_i, m') = (P_i, \ldots, P_d)$.*

ii. ***Correct return path.***

- $\mathcal{P}^{\leftarrow}(H_1, C_1, P_1, m') = (P_{d+1}, \ldots, P_s)$.
- *For every $i \in \{d+1, \ldots, s\}$, reply message m'' and content C such that $|C| = |C_i|$, $\mathcal{P}^{\rightarrow}(H_i, C, P_i, m'') = (P_{d+1}, \ldots, P_s)$.*

iii. ***Correct layering.*** $\mathcal{L}(H_1, C_1, P_1, m') = (H_1, C_1, \ldots, H_d, C_d, H_{d+1}, \ldots, H_s)$,

iv. ***Correct message.*** $\mathsf{ProcOnion}((H_d, C_d), P_d, \mathsf{sk}(P_d)) = (\mathsf{R}, m)$,

v. ***Correct reply message.*** $\mathsf{ProcOnion}((H_s, C_s), P_s, \mathsf{sk}(P_s)) = (\mathsf{S}, (\ell, m'))$ *where (H_s, C_s) are the header and content of the last onion in the evolution $\mathcal{E}(H_1, C_1, P_1, m')$.*

Remark We define onion evolution, (forward and return) paths and layering so that we can articulate what it means for an onion encryption scheme to be correct. We define correctness to mean that how an onion peels (the evolution, paths and layerings) exactly reflects the reverse process of how the onion was built up. Thus, for our definition to make sense, both ProcOnion and FormReply must be deterministic processes given the secret keys.

4.3 $\mathcal{F}_{\mathsf{ROES}}$: onion routing in the SUC Framework

In this section, we provide a formal definition of security for repliable onion encryption schemes. We chose to define security in the simplified universal composability (SUC) model [CCL15] as opposed to the universal composability (UC) model [Can01] as this choice greatly simplifies how communication is modeled, in turn, allowing for a more easily digestible description of the ideal functionality. Additionally, since SUC-realizability implies UC-realizability [CCL15], we do not lose generality by simplifying the model in this manner.

Communication model In the SUC model, the environment $\mathcal{Z}$ can communicate directly with each party P by writing inputs into P's input tape and by reading P's output tape. The parties communicate with each other and also with the ideal functionality through an additional party, the router $\mathcal{R}$.

We first describe the ideal functionality $\mathcal{F}_{\mathsf{ROES}}$ (ROES, for "repliable onion encryption scheme") for the repliable onion encryption scheme. See Figure 4.1 for a summary of $\mathcal{F}_{\mathsf{ROES}}$.

IdealSetup

1: Get from ideal adversary $\mathcal{A}$: $\mathcal{P}$, Bad, G, ProcOnion, FormReply, SampleOnion, CompleteOnion, RecoverReply

2: Initialize dictionaries OnionDict and PathDict

IdealFormOnion$(\ell, m, P^{\rightarrow}, P^{\leftarrow})$

1: Break forward path into segments

2: Run SampleOnion on segments to generate onion layers

3: Store onion layers in OnionDict

4: Store label and (rest of) return path in PathDict

IdealProcOnion$((H, C), P)$

1: If (P, H) is "familiar"

- If (P, H, C) in OnionDict, return stored
- Else if exists $(P, H, (X \neq C))$ in OnionDict, return output of CompleteOnion and stored party, or "$\perp$"
- Else if $(P, H, \star)$ in PathDict, return output of IdealFormOnion on message recovered using RecoverReply and label and path stored in PathDict

2: Else if (P, H) is not familiar, return output of ProcOnion

IdealFormReply$(m, (H, C), P)$

1: If (P, H, C) in PathDict, return output of IdealFormOnion on m and label and path stored in PathDict

2: Else, return output of FormReply

Figure 4.1: Summary of ideal functionality $\mathcal{F}_{\mathsf{ROES}}$.

4.3.1 Ideal functionality $\mathcal{F}_{\mathsf{ROES}}$

Notation In this section, honest parties are capitalized, e.g., P, P_i; and corrupt parties are generally written in lowercase, e.g., p, p_i. An onion formed by an honest party is *honestly formed* and is capitalized, e.g., O, O_i; whereas, an onion formed by a corrupt party is generally written in lowercase, e.g., o, o_i. Recall that an onion O is a pair, consisting of the (encrypted) content C and the header H, i.e., $O = (H, C)$.

The ideal functionality $\mathcal{F}_{\mathsf{ROES}}$ handles requests from the environment (to form an onion, process an onion or form a return onion) on behalf of the ideal honest parties.

Setting up.

Each static setting for a fixed set of participants and a fixed public key infrastructure requires a separate setup. During setup, $\mathcal{F}_{\mathsf{ROES}}$ gets the following from the ideal adversary $\mathcal{A}$. (For each algorithm in items (iv)-(vi), we first describe the input of the algorithm in normal font and then, in italics, provide a brief preview of how the algorithm will be used. $\mathcal{F}_{\mathsf{ROES}}$ only

runs for a polynomial number of steps which is specified in the public parameters pp and can time out on running these algorithms from the ideal adversary.)

i. the set $\mathcal{P}$ of participants

ii. the set Bad of corrupt parties in $\mathcal{P}$

iii. the repliable onion encryption scheme's G, $\mathsf{ProcOnion}$ and $\mathsf{FormReply}$ algorithms:

- *G is used for generating the honest parties' keys.*
- $\mathsf{ProcOnion}$ *is used for processing onions formed by corrupt parties.*
- $\mathsf{FormReply}$ *is used for replying to onions formed by corrupt parties.*

iv. the p.p.t. algorithm $\mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, p^\rightarrow, p^\leftarrow, m)$ that takes as input the security parameter 1^λ, the public parameters pp, the forward path $p^\rightarrow$, the (possibly empty) return path $p^\leftarrow$ and the (possibly empty) message m. The routing path $(p^\rightarrow, p^\leftarrow) = (p_1, \ldots, p_i, P_{i+1})$ is always a sequence $(p_1, \ldots, p_i)$ of adversarial parties, possibly ending in an honest party P_{i+1}. $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ fails if $\mathsf{SampleOnion}$ ever samples a repeating header or key.

$\mathsf{SampleOnion}$ *is used to compute an onion to send to p_1 which will be "peelable" all the way to an onion for P_{i+1}. If the return path $p^\leftarrow$ is non-empty and ends in an honest party P_{i+1},* $\mathsf{SampleOnion}$ *produces an onion o for the first party p_1 in $p^\rightarrow$ and a header H for the last party P_{i+1} in $p^\leftarrow$. Else if the return path $p^\leftarrow$ is empty, and the forward path $p^\rightarrow$ ends in an honest party P_{i+1},* $\mathsf{SampleOnion}$ *produces an onion o for the first party p_1 in $p^\rightarrow$ and an onion O for the last party P_{i+1} in $p^\rightarrow$. Else if the return path $p^\leftarrow$ is empty, and the forward path $p^\rightarrow$ ends in a corrupt party p_i,* $\mathsf{SampleOnion}$ *produces an onion o for the first party p_1 in $p^\rightarrow$.*

v. the p.p.t. algorithm $\mathsf{CompleteOnion}(1^\lambda, \mathsf{pp}, H', C)$ that takes as input the security parameter 1^λ, the public parameters pp, the identity of the party P, the header H' and the content C, and outputs an onion $O = (H', C')$. $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ fails if $\mathsf{CompleteOnion}$ ever produces a repeating onion.

$\mathsf{CompleteOnion}$ *produces an onion (H', C') that resembles the result of peeling an onion with content C.*

vi. the d.p.t. algorithm $\mathsf{RecoverReply}(1^\lambda, \mathsf{pp}, O, P)$ that takes as input the security parameter 1^λ, the public parameters pp, the onion O and the party P, and outputs a label ℓ and a reply message m.

This algorithm is used for recovering the label ℓ and reply message m from the return onion O that carries the response from a corrupt recipient to an honest sender.

$\mathcal{F}$ creates a copy $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ of itself for handling instructions pertaining to session-id sid and sends items (i)-(vi) to $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$; these items pertain to the same static public key infrastructure setting.

$\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ generates a public key pair $(\mathsf{pk}(P), \mathsf{sk}(P))$ for each honest party $P \in \mathcal{P} \setminus \mathsf{Bad}$ using the key generation algorithm G and sends the public keys to their respective party. (If working within the global PKI framework, each party then relays his/her key to the global bulletin board functionality [CSV16].)

$\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ also creates the following (initially empty) dictionaries:

- The *onion dictionary* $\mathsf{OnionDict}$ supports:
  - a method $\mathsf{put}((P, H, C), (\mathsf{role}, \mathsf{output}))$ that stores under the label (P, H, C): the role "role" and the output "output"
  - a method $\mathsf{lookup}(P, H, C)$ that looks up the entry $(\mathsf{role}, \mathsf{output})$ corresponding to the label (P, H, C)
- The *return path dictionary* $\mathsf{PathDict}$ supports:
  - a method $\mathsf{put}((P, H, C), (P^{\leftarrow}, \ell))$ that stores under the label (P, H, C): the return path $P^{\leftarrow}$ and the label ℓ
  - a method $\mathsf{lookup}(P, H, C)$ that looks up the entry $(P^{\leftarrow}, \ell)$ corresponding to the label (P, H, C)

These data structures are stored internally at and are accessible only by $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$.

Forming an onion.

After setup, the environment $\mathcal{Z}$ can instruct an honest party P to form an onion using the session-id sid, the label ℓ, the message m, the forward path $P^{\rightarrow}$ and the return path $P^{\leftarrow}$. To form the onion, P forwards the instruction from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ (via the router $\mathcal{R}$).

The goal of the ideal functionality $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ is to create and maintain state information for handling an onion O (the response to the "form onion" request). O should be "peelable" by the parties in the forward path $P^{\rightarrow}$, internally associated with the return path $P^{\leftarrow}$, and for the purpose of realizing this functionality by an onion encryption scheme, each layer of

the onion should look "believable" as onions produced from running FormOnion, ProcOnion or FormReply.

Importantly, O and its onion layers should reveal no information to $\mathcal{A}$, by which we mean the following:

- Each onion routed to an honest party P_i is formed initially with just (P_i) as the routing path and, therefore, reveals only that it is for P_i. When forming the onion, no message is part of the input; this ensures that the onion is information-theoretically independent of any message m.

- For every party p_i or P_i in the forward path, let $\mathsf{next}(i)$ denote the index of the next honest party $P_{\mathsf{next}(i)}$ following p_i. For example, if the forward path is $(P_1, p_2, p_3, P_4, P_5, p_6, p_7)$, then $\mathsf{next}(2) = 4$.

  Conceptually, each onion routed to an adversarial party p_i is formed by "wrapping" an onion layer for each corrupt party in $(p_i, \ldots, p_{\mathsf{next}(i)-1})$ (or $(p_{i+1}, \ldots, p_{|P^{\rightarrow}|})$ if no honest party after p_i exists) around an onion formed for an honest party $P_{\mathsf{next}(i)}$ (or a message if no honest party after p_i exists). This reveals at most the sequence $(p_i, \ldots, p_{\mathsf{next}(i)-1}, P_{\mathsf{next}(i)})$ (or the sequence $(p_i, \ldots, p_{|P^{\rightarrow}|})$ and the message m if no honest party after p_i exists). How this wrapping occurs depends on the internals of the SampleOnion algorithm provided by the ideal adversary.

To ensure these properties, the ideal functionality partitions the forward path $P^{\rightarrow}$ into segments, where each segment starts with a sequence of corrupt parties and ends with a single honest party:

Let P_f (f, for first) be the first honest party in the forward path. The first couple of segments are $(p_1, \ldots, p_{f-1}, P_f)$, $(p_{f+1}, \ldots, p_{\mathsf{next}(f)-1}, P_{\mathsf{next}(f)})$, etc.

For each segment $(p_i, \ldots, p_{j-1}, P_j)$, the ideal functionality $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ samples onions (h_i, c_i) and (H_j, C_j) using the algorithm SampleOnion, i.e., $((h_i, c_i), (H_j, C_j)) \leftarrow \mathsf{SampleOnion}(1^{\lambda}, \mathsf{pp}, (p_i, \ldots, p_{j-1}, P_j), (), \perp)$.

Let P_e (e, for end) be the last honest party in the forward path $P^{\rightarrow}$, and let $P_{\mathsf{next}(d)}$ denote the first honest party in the return path $P^{\leftarrow}$. If the recipient p_d is corrupt, then $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ also runs $\mathsf{SampleOnion}(1^{\lambda}, \mathsf{pp}, (p_{e+1}, \ldots, p_d), (p_{d+1}, \ldots, p_{\mathsf{next}(d)-1}, P_{\mathsf{next}(d)}), m)$; this produces an onion o_{e+1} and a header $H_{\mathsf{next}(d)}$.

For every honest intermediary party P_i in the forward path, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ stores under the label (P_i, H_i, C_i) in the onion dictionary OnionDict the role "I", the $(i+1)$-st onion layer (H_{i+1}, C_{i+1}) and destination P_{i+1}. The $(d+1)$-st onion layer doesn't exist for the innermost

layer (H_d, C_d) for an honest recipient P_d. In this case, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ stores just the role "R" and the message m.

If the recipient P_d is honest, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ stores the entry $((P_d, H_d, C_d), (P^{\leftarrow}, \ell))$ in the dictionary PathDict. Otherwise if the recipient p_d is corrupt, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ stores the entry $((P_{\mathsf{next}(d)}, H_{\mathsf{next}(d)}, *), (p^{\leftarrow}, \ell))$ in PathDict where $p^{\leftarrow} = (p_{\mathsf{next}(d)+1}, \ldots, P_s)$. ("$*$" is the unique symbol that means "any content".)

$\mathsf{IdealFormOnion}(\ell, m, (P_1, \ldots, P_d), (P_{d+1}, \ldots, P_s))$

1 : $\mathsf{cur} = 1$
2 : $\mathsf{next} = \mathsf{next}(\mathsf{cur})$
3 : **while** $\mathsf{next} \leq d$
4 : $\quad p^{\rightarrow} = (p_{\mathsf{cur}+1}, \ldots, p_{\mathsf{next}-1}, P_{\mathsf{next}})$
5 : $\quad (o_{\mathsf{cur}+1}, O_{\mathsf{next}}, \kappa_{\mathsf{next}}) \leftarrow \mathsf{SampleOnion}(1^{\lambda}, \mathsf{pp}, p^{\rightarrow}, (), \perp)$
6 : $\quad \mathsf{cur} = \mathsf{next}$
7 : $\quad \mathsf{next} = \mathsf{next}(\mathsf{cur})$
8 : **if** the recipient p_d is corrupt
9 : $\quad p^{\rightarrow} = (p_{\mathsf{cur}+1}, \ldots, p_d)$
10 : $\quad p^{\leftarrow} = (p_{d+1}, \ldots, p_{\mathsf{next}(d)-1}, P_{\mathsf{next}(d)})$
11 : $\quad (o_{\mathsf{cur}+1}, H_{\mathsf{next}(d)}, \kappa_{\mathsf{next}(d)}) \leftarrow \mathsf{SampleOnion}(1^{\lambda}, \mathsf{pp}, p^{\rightarrow}, p^{\leftarrow}, m)$
12 : $\mathsf{cur} = \mathsf{next}(1)$
13 : **while** $\mathsf{cur} < d$
14 : $\quad \mathsf{OnionDict.put}((P_{\mathsf{cur}}, H_{\mathsf{cur}}, C_{\mathsf{cur}}), (\mathsf{I}, ((H_{\mathsf{cur}+1}, C_{\mathsf{cur}+1}), P_{\mathsf{cur}+1}), \kappa_{\mathsf{cur}+1})$
15 : **if** the recipient P_d is honest
16 : $\quad \mathsf{OnionDict.put}((P_d, H_d, C_d), (\mathsf{R}, m), \perp)$
17 : $\quad \mathsf{PathDict.put}((P_d, H_d, C_d), (P^{\leftarrow}, \ell))$
18 : **else if** $\mathsf{next}(d) \leq s$
19 : $\quad ((P_{\mathsf{next}(d)}, H_{\mathsf{next}(d)}, *), ((p_{\mathsf{next}(d)+1}, \ldots, P_s, \ell))$
20 : **return** (H_1, C_1)

Figure 4.2: Pseudocode for $\mathcal{F}_{\mathsf{ROES}}$'s "onion forming" algorithm IdealFormOnion. On input the label ℓ, the message m, the forward path $(P_1, \ldots, P_d)$ and the return path $(P_{d+1}, \ldots, P_s)$, IdealFormOnion outputs an onion (H_1, C_1). When IdealFormOnion forms onions for the return path, it outputs string "S" in place of "R" (in line 19).

See Figure 4.2 for the pseudocode for $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$'s "onion forming" algorithm.

Example 1. The recipient P_7 is honest. The forward path is $P^{\rightarrow} = (P_1, p_2, p_3, P_4, P_5, p_6, \boxed{P_7})$, and the return path is $P^{\leftarrow} = (p_8, p_9, P_{10}, p_{11}, P_{12})$. In this case,

the first segment is (P_1), and the second segment is (p_2, p_3, P_4) and so on; and

$$\begin{aligned}
(\bot, (H_1, C_1)) &\leftarrow \mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, (P_1), (), \bot) \\
((h_2, c_2), (H_4, C_4)) &\leftarrow \mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, (p_2, p_3, P_4), (), \bot) \\
(\bot, (H_5, C_5)) &\leftarrow \mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, (P_5), (), \bot) \\
((h_6, c_6), (H_7, C_7)) &\leftarrow \mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, (p_6, P_7), (), \bot).
\end{aligned}$$

$\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ stores in $\mathsf{OnionDict}$:

$$\begin{aligned}
&\mathsf{OnionDict.put}((P_1, H_1, C_1), (\mathsf{I}, ((h_2, c_2), p_2))) \\
&\mathsf{OnionDict.put}((P_4, H_4, C_4), (\mathsf{I}, ((H_5, C_5), P_5))) \\
&\mathsf{OnionDict.put}((P_5, H_5, C_5), (\mathsf{I}, ((h_6, c_6), p_6))) \\
&\mathsf{OnionDict.put}((P_7, H_7, C_7), (\mathsf{R}, m)),
\end{aligned}$$

and stores in $\mathsf{PathDict}$:

$$\mathsf{PathDict.put}((P_7, H_7, C_7), ((p_8, p_9, P_{10}, p_{11}, P_{12}), \ell)).$$

Example 2. The recipient p_7 is corrupt. The forward path is $P^{\rightarrow} = (P_1, p_2, p_3, P_4, P_5, p_6, \boxed{p_7})$, and the return path is $P^{\leftarrow} = (p_8, p_9, P_{10}, p_{11}, P_{12})$. In this case,

$$\begin{aligned}
(\bot, (H_1, C_1)) &\leftarrow \mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, (P_1), (), \bot) \\
((h_2, c_2), (H_4, C_4)) &\leftarrow \mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, (p_2, p_3, P_4), (), \bot) \\
(\bot, (H_5, C_5)) &\leftarrow \mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, (P_5), (), \bot) \\
(o_6, H_{10}) &\leftarrow \mathsf{SampleOnion}(1^\lambda, \mathsf{pp}, (p_5, p_6, p_7), (p_8, p_9, P_{10}), m).
\end{aligned}$$

$\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ stores in $\mathsf{OnionDict}$:

$$\begin{aligned}
&\mathsf{OnionDict.put}((P_1, H_1, C_1), (\mathsf{I}, ((h_2, c_2), p_2))) \\
&\mathsf{OnionDict.put}((P_4, H_4, C_4), (\mathsf{I}, ((H_5, C_5), P_5))) \\
&\mathsf{OnionDict.put}((P_5, H_5, C_5), (\mathsf{I}, ((h_6, c_6), p_6))),
\end{aligned}$$

and stores in $\mathsf{PathDict}$:

$$\mathsf{PathDict.put}((P_{10}, H_{10}, *), ((p_{11}, P_{12}), \ell)).$$

After updating $\mathsf{OnionDict}$ and $\mathsf{PathDict}$, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ returns the first onion $O_1 = (H_1, C_1)$ to party P (via the router $\mathcal{R}$). Upon receiving O_1 from $\mathcal{F}$, P outputs the session id sid and O_1.

Processing an onion.

After setup, the environment $\mathcal{Z}$ can instruct an honest party P to process an onion $O = (H, C)$ for the session-id sid. To process the onion, party P forwards the instruction to the ideal functionality $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ (via the router $\mathcal{R}$).

The ideal functionality first checks if there is an entry under (P, H, C) in the dictionary $\mathsf{OnionDict}$.

Case 1 There is an entry $(\mathsf{role}, \mathsf{output})$ under the label (P, H, C) in $\mathsf{OnionDict}$. In this case, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ responds to P (via the router $\mathcal{R}$) with $(\mathsf{role}, \mathsf{output})$.

Case 2 There is no entry under the label (P, H, C) in $\mathsf{OnionDict}$, but there exists $X \neq C$ such that there is an entry $(\mathsf{I}, ((H', X'), P'), k)$ under the label (P, H, X) in $\mathsf{OnionDict}$. In this case, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ samples an onion $(H', C') \leftarrow \mathsf{CompleteOnion}(1^\lambda, \mathsf{pp}, H', C)$, stores the new entry $(\mathsf{I}, ((H', C'), P'))$ under the label (P, H, C) in $\mathsf{OnionDict}$ and responds to P with $(\mathsf{I}, ((H', X'), P'))$.

Case 3 There is no entry under the label (P, H, C) in $\mathsf{OnionDict}$, but there exists $X \neq C$ such that there is an entry (R, m) under the label (P, H, X) in $\mathsf{OnionDict}$. In this case, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ responds to P with $(\mathsf{R}, \perp)$.

Case 4 There is no entry under the label (P, H, C) in $\mathsf{OnionDict}$, but there exists $X \neq C$ such that there is an entry $(\mathsf{S}, (\ell, m))$ under the label (P, H, X) in $\mathsf{OnionDict}$. In this case, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ responds to P with $(\mathsf{S}, \perp)$.

Case 5 There is no entry starting with (P, H) in $\mathsf{OnionDict}$, but there is an entry $(P^{\leftarrow}, \ell)$ under the label $(P, H, *)$ in $\mathsf{PathDict}$. Let m' be the message obtained from running $\mathsf{RecoverReply}(1^\lambda, \mathsf{pp}, O, P)$.

If $P^{\leftarrow}$ is not empty, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ runs its "form onion" code ($\mathsf{IdealFormOnion}$ in Figure 4.2) with (ℓ, m') as the "message", $P^{\leftarrow}$ as the forward path and the empty list "()" as the return path. ($\mathsf{IdealFormOnion}$ is run with auxiliary information for correctly labeling the last party in $P^{\leftarrow}$ as the sender.) In this case, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ responds to P with $(\mathsf{I}, ((H', C'), P'))$, where (H', C') is the returned onion, and P' is the first party in $P^{\leftarrow}$.

Otherwise if $P^{\leftarrow}$ is empty, then P is the recipient of the return onion, so $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ responds to P with $(\mathsf{S}, (\ell, m'))$.

Case 6 $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ doesn't know how to peel O (i.e., there is no entry starting with (P, H) in $\mathsf{OnionDict}$ and no entry under $(P, H, *)$ in $\mathsf{PathDict}$). In this case, O does not have an honestly formed header; so, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ responds to P with $(\mathsf{role}, \mathsf{output}) = \mathsf{ProcOnion}(1^\lambda, \mathsf{pp}, O, P, \mathsf{sk}(P))$.

Upon receiving the response $(\mathsf{role}, \mathsf{output})$ from $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$, P outputs the session id sid and $(\mathsf{role}, \mathsf{output})$.

Forming a reply.

After setup, the environment $\mathcal{Z}$ can instruct an honest party P to form a reply using the session-id sid, the reply message m and an onion $O = (H, C)$. To form the return onion, P forwards the instruction to the ideal functionality $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ (via the router $\mathcal{R}$).

Upon receiving the forwarded request, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ looks up (P, H, C) in $\mathsf{PathDict}$.

Case 1 There is an entry $(P^{\leftarrow}, \ell)$ under the label (P, H, C) in $\mathsf{PathDict}$. Then $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ runs its "form onion" code ($\mathsf{IdealFormOnion}$ in Figure 4.2) with (ℓ, m) as the "message", $P^{\leftarrow}$ as the forward path and the empty list "()" as the return path. ($\mathsf{IdealFormOnion}$ is run with auxiliary information for correctly labeling the last party in $P^{\leftarrow}$ as the sender.) $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ responds to P (via the router $\mathcal{R}$) with the returned onion O' and the first party P' in $P^{\leftarrow}$.

Case 2 No entry exists for (P, H, C) in $\mathsf{PathDict}$. Then P is replying to an onion formed by an adversarial party, so $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ replies to P with $(O', P') = \mathsf{FormReply}(1^\lambda, \mathsf{pp}, m, O, P, \mathsf{sk}(P))$. Upon receiving the response (O', P') from $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$, P outputs the session id sid and (O', P').

4.3.2 SUC-realizability of $\mathcal{F}_{\mathsf{ROES}}$

Ideal protocol In the ideal onion routing protocol, the environment $\mathcal{Z}$ interacts with the participants by writing instructions into the participants' input tapes and reading the participants' output tapes. Each input is an instruction to form an onion, process an onion or form a return onion. When an honest party P receives an instruction from $\mathcal{Z}$, it forwards the instruction to the ideal functionality $\mathcal{F}_{\mathsf{ROES}}$ via the router $\mathcal{R}$. Upon receiving a response from $\mathcal{F}_{\mathsf{ROES}}$ (via $\mathcal{R}$), P outputs the response. Corrupt parties are controlled by the adversary $\mathcal{A}$ and behave according to $\mathcal{A}$. $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ does not interact with $\mathcal{A}$ after the setup phase.

At the end of the protocol execution, $\mathcal{Z}$ outputs a bit b. Let $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{ROES}}, \mathcal{A}, \mathcal{Z}}(1^\lambda, \mathsf{pp})$ denote $\mathcal{Z}$'s output after executing the ideal protocol for security parameter 1^λ and public parameters pp.

Real protocol Let Σ be a repliable onion encryption scheme. The real onion routing protocol for Σ is the same as the ideal routing protocol (described above), except that the honest parties simply run Σ's algorithms to form and process onions.

Let $\mathsf{REAL}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda, \mathsf{pp})$ denote $\mathcal{Z}$'s output after executing the real protocol.

Definition 15 (SUC-realizability of $\mathcal{F}_{\mathsf{ROES}}$)**.** *The repliable onion encryption scheme* Σ SUC-realizes *the ideal functionality* $\mathcal{F}_{\mathsf{ROES}}$ *if for every p.p.t. real-model adversary* $\mathcal{A}$*, there exists a p.p.t. ideal-model adversary* $\mathcal{S}$ *such that for every polynomial-time balanced environment* $\mathcal{Z}$*, there exists a negligible function* $\nu(\lambda)$ *such that*

$$\left|\Pr\left[\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{ROES}},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathsf{pp}) = 1\right] - \Pr\left[\mathsf{REAL}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda, \mathsf{pp}) = 1\right]\right| \leq \nu(\lambda).$$

4.3.3 Remarks

On the assumption that keys are consistent with PKI In describing the ideal functionality, we made an implicit assumption that for every instruction to form an onion, the keys match the parties on the routing path. However, generally speaking, the environment $\mathcal{Z}$ can instruct an honest party to form an onion using the wrong keys for some of the parties on the routing path. Using the dictionary $\mathsf{OnionDict}$, it is easy to extend our ideal functionality to cover this case: the ideal functionality would store in $\mathsf{OnionDict}$, every onion layer for an honest party, starting from the outermost layer, until it reaches a layer with a mismatched key. To keep the exposition clean, we will continue to assume that inputs are well-behaved, i.e., router names are valid, and keys are as published.

On replayed onions As originally noted by Camenisch and Lysyanskaya [CL05], the environment is allowed to repeat the same input (e.g., a "process onion" request) in the UC framework (likewise, in the SUC framework). Thus, replay attacks are not only allowed in our model but inherent in the SUC framework. The reason that replay attacks are a concern is that they allow the adversary to observe what happens in the network as a result of repeatedly sending an onion over and over again — which intermediaries are involved, etc — and that potentially allows the adversary to trace this onion. Our functionality does not protect from this attack (and neither did the CL functionality), but a higher-level protocol can address this by directing parties to ignore repeat "process onion" and "form reply" requests. Other avenues to address this (which can be added to our functionality, but we chose not to so as not to complicate it further) may include letting onions time out, so the time frame for repeating them could be limited.

4.4 Shallot Encryption: our repliable onion encryption scheme

In this section, we provide our construction of a repliable onion encryption scheme dubbed "*Shallot Encryption Scheme*". Like in Camenisch and Lysyanskaya's construction [CL05], each onion layer for a party P is encrypted under a key k which, in turn, is encrypted under the public key of P and a tag t. Our construction differs from the original construction in that the tag t is not a function of the layer's content. Instead, authentication of the message happens separately, using a message authentication code (MAC). The resulting object is more like a shallot than an onion; it consists of *two* layered encryption objects: the header and the content (which may contain a "bud", i.e., another layered encryption object, namely the header for the return onion). We still call these objects "onions" to be consistent with prior work, but the scheme overall merits the name "shallot encryption".

Notation Let λ denote the security parameter. Let $\{F_{\mathsf{seed}(\cdot,\cdot)}\}_{\mathsf{seed}\in\{0,1\}^*}$ be a pseudorandom function (PRF) of two inputs. Let $\{f_k(\cdot)\}_{k\in\{0,1\}^*}$ and $\{g_k(\cdot)\}_{k\in\{0,1\}^*}$ be block ciphers, i.e., pseudorandom permutations (PRPs). We use the same key to key both block ciphers: one ($\{f_k(\cdot)\}_{k\in\{0,1\}^*}$) with a "short" blocklength $L_1(\lambda)$ is used for forming headers, the other ($\{g_k(\cdot)\}_{k\in\{0,1\}^*}$) with a "long" blocklength $L_2(\lambda)$ is used for forming contents. This is standard and can be constructed from regular block ciphers. Following the notational convention introduced by Camenisch and Lysyanskaya [CL05], let $\{X\}_k$ denote $f_k(X)$ (or $g_k(X)$) , and let $\}X\{_k$ denote $f_k^{-1}(X)$ (or $g_k^{-1}(X)$).

Let $\mathcal{E} = (\mathsf{Gen}_{\mathcal{E}}, \mathsf{Enc}, \mathsf{Dec})$ be a CCA2-secure encryption scheme with tags [CS98], let $\mathsf{MAC} = (\mathsf{Gen}_{\mathsf{MAC}}, \mathsf{Tag}, \mathsf{Ver})$ be a MAC, and let h be a collision-resistant hash function.

4.4.1 Setting up

Each party P_i forms a public key pair $(\mathsf{pk}(P_i), \mathsf{sk}(P_i))$ using the public key encryption scheme's key generation algorithm $\mathsf{Gen}_{\mathcal{E}}$, i.e., $(\mathsf{pk}(P_i), \mathsf{sk}(P_i)) \leftarrow \mathsf{Gen}_{\mathcal{E}}(1^\lambda, \mathsf{pp}, P_i)$.

4.4.2 Forming a repliable onion

Each onion consists of (1) the header (i.e., the encrypted routing path and associated keys) and (2) the content (i.e., the encrypted message).

Forming the header In our example, let Anna (denoted P_s) be the sender, and let Roberto (denoted P_d, d for destination) be the recipient. To form a *repliable onion*, Anna

receives as input a label ℓ, a message m, a *forward* path to Roberto

$$P^{\rightarrow} = P_1, \ldots, P_{d-1}, P_d, \qquad |P^{\rightarrow}| = d \leq N,$$

and a *return* path to herself:

$$P^{\leftarrow} = P_{d+1}, \ldots, P_{s-1}, P_s, \qquad |P^{\leftarrow}| = s - d + 1 \leq N.$$

In other words, Roberto is P_d, and Anna is P_s. All other participants P_i are intermediaries.

Let "seed" be a seed stored in $\mathsf{sk}(P_s)$. Anna computes (i) an encryption key $k_i = F_{\mathsf{seed}}(\ell, i)$ for every party P_i on the routing path $(P^{\rightarrow}, P^{\leftarrow})$, (ii) an authentication key K_d for Roberto using $\mathsf{Gen}_{\mathsf{MAC}}(1^\lambda)$ with $F_{\mathsf{seed}}(d, \ell)$ sourcing the randomness for running the key generation algorithm and (iii) an authentication key K_s for herself using $\mathsf{Gen}_{\mathsf{MAC}}(1^\lambda)$ with $F_{\mathsf{seed}}(s, \ell)$ sourcing the randomness for running the key generation algorithm.

Remark: We can avoid using a PRF in exchange for requiring state; an alternative to using a PRF is to store keys computed from true randomness locally, e.g., in a dictionary.

The goal of $\mathsf{FormOnion}$ is to produce an onion O_1 for the first party P_1 on the routing path such that P_1 processing O_1 produces the onion O_2 for the next destination P_2 on the routing path, and so on.

Suppose for the time being that $d = s - d + 1 = N$.

Let O be an onion peelable by party P. The *header* of O is a sequence $H = (E, B^1, \ldots, B^{N-1})$. E is an encryption under the tag $t = h(B^1, \ldots, B^{N-1})$ of (i) P's role, (ii) P's encryption key k (or label ℓ) and (iii) authentication key K (if it exists). The decryption $\}B^1\{_k$ reveals the destination P' and the ciphertext E' of the peeled onion. For each $1 < j < N$, the decryption $\}B^j\{_k$ is block $(B')^{j-1}$ of the peeled onion, so the header of the peeled onion will begin with $(E', (B')^1, \ldots, (B')^{N-2})$. The final block $(B')^{N-1}$ of the header is formed by "decrypting" the all-zero string of length $L_1(\lambda)$, i.e., $(B')^{N-1} = \}0 \ldots 0\{_k$.

Anna first forms the header $H_d = (E_d, B_d^1, \ldots, B_d^{N-1})$ for the last onion O_d on the forward path (the one to be processed by Roberto): For every $i \in \{1, \ldots, N-1\}$, let

$$B_d^i = \} \ldots \} 0 \ldots 0 \{_{k_i} \ldots \{_{k_{d-1}}.$$

The tag t_d for integrity protection is the hash of these blocks concatenated together, i.e.,

$$t_d = h(B_d^1, \ldots, B_d^{N-1}).$$

The ciphertext E_d is the encryption of (R, k_d, K_d) under the public key $\mathsf{pk}(P_d)$ and the tag t_d, i.e.,

$$E_d \leftarrow \mathsf{Enc}(\mathsf{pk}(P_d), t_d, (\mathsf{R}, k_d, K_d)).$$

The headers of the remaining onions in the evolution are formed recursively. Let

$$\begin{aligned}
B^1_{d-1} &= \{P_d, E_d\}_{k_{d-1}}, \\
B^i_{d-1} &= \{B^{i-1}_d\}_{k_{d-1}}, && \forall i \in \{2, \ldots, N-1\}, \\
t_{d-1} &= h(B^1_{d-1}, \ldots, B^{N-1}_{d-1}), \\
E_{d-1} &\leftarrow \mathsf{Enc}(\mathsf{pk}(P_{d-1}), t_{d-1}, (\mathsf{I}, k_{d-1}));
\end{aligned}$$

and so on. (WLOG, we assume that (P_d, E_d) "fits" into a block; i.e., $|P_d, E_d| \leq L_1(\lambda)$. A block cipher with the correct blocklength can be built from a standard one [KL14, BS15].) See FormHeader in Figure 4.3.

Forming the encrypted content Anna then forms the encrypted content for Roberto.

First, if the return path $P^{\leftarrow}$ is non-empty, Anna forms the header H_{d+1} for the return onion using the same procedure that she used to form the header H_1 for the forward onion, but using the return path $P^{\leftarrow}$ instead of the forward path $P^{\rightarrow}$ and encrypting (S, ℓ) instead of (R, k_s, K_s). That is, the ciphertext E_s of the "innermost" header H_s is the encryption $\mathsf{Enc}(\mathsf{pk}(P_s), t_s, (\mathsf{S}, \ell))$ rather than $\mathsf{Enc}(\mathsf{pk}(P_s), t_s, (\mathsf{R}, k_s, K_s))$. If the return path is empty, then H_{d+1}, k_s and K_s are the empty string.

When Roberto processes the onion, Anna wants him to receive (i) the message m, (ii) the header H_{d+1} for the return onion, (iii) the keys k_s and K_s for forming a reply to the anonymous sender (Anna) and (iv) the first party P_{d+1} on the return path. So, Anna sets the "*meta-message*" M to be the concatenation of m, H_{d+1}, k_s, K_s and P_{d+1}: $M = (m, H_{d+1}, k_s, K_s, P_{d+1})$.

Anna wants Roberto to be able to verify that M is the meta-message, so she also computes the tag $\sigma_d = \mathsf{Tag}(K_d, M)$. (WLOG, (M, σ_d) "fits" exactly into a block; i.e., $|M| \leq L_2(\lambda)$.)

The encrypted content C_i for each onion O_i on the forward path is given by:

$$C_i = \{\ldots\{M, \sigma_d\}_{k_d}\ldots\}_{k_i};$$

see FormContent in Figure 4.3.

We now explain what happens when $d \neq N$, or $s - d + 1 \neq N$:

If either d or $s - d + 1$ exceed the upper bound N, then FormOnion returns an error. If d is strictly less than N, the header is still "padded" to $N - 1$ blocks by sampling N encryption keys as before. Likewise if $s - d + 1 < N$, the header is padded to $N - 1$ blocks in similar fashion.

4.4.3 Processing a repliable onion (in the forward direction)

Let Carol be an intermediary node on the forward path from Anna to Roberto. When Carol receives the onion $O_i = (H_i, C_i)$ consisting of the header $H_i = (E_i, B_i^1, \ldots B_i^{N-1})$ and the content C_i, she processes it as follows:

Carol first computes the tag $t_i = h(B_i^1, \ldots B_i^{N-1})$ for integrity protection and then attempts to decrypt the ciphertext E_i of the header using her secret key $\mathsf{sk}(P_i)$ and the tag t_i to obtain her role and key(s), i.e.,

$$(\mathsf{I}, k_i) = \mathsf{Dec}(\mathsf{sk}(P_i), t_i, E_i).$$

Carol succeeds in decrypting E_i only if the header has not been tampered with. In this case, she gets her role "I" and the key k_i and proceeds with processing the header and content:

Carol first decrypts the first block B_i^1 of the current header to retrieve the next destination P_{i+1} and ciphertext E_{i+1} of the processed header (header of the next onion), i.e.,

$$(P_{i+1}, E_{i+1}) = \}B_i^1\{_{k_i}.$$

To obtain the first $N-2$ blocks of the processed header, Carol decrypts the last $N-2$ blocks of H:

$$B_{i+1}^j = \}B_i^{j+1}\{_{k_i} \qquad \forall j \in [N-2].$$

To obtain the last block of the processed header, Carol "decrypts" the all-zero string "$(0 \ldots 0)$":

$$B_{i+1}^{N-1} = \}0 \ldots 0\{_{k_i}.$$

To process the content, Carol simply decrypts the current content C_i:

$$C_{i+1} = \}C_i\{_{k_i}.$$

See ProcOnion in Figure 4.4.

4.4.4 Replying to the anonymous sender

When Roberto receives the onion $O_d = (H_d, C_d)$, he processes it in the same way that the intermediary party Carol does, by running ProcOnion:

Roberto first decrypts the ciphertext E_d of the header to retrieve his role "R" and the keys k_d and K_d. If O_d hasn't been tampered with, Roberto retrieves the meta-message $M =$

$(m, H_{d+1}, k_s, K_s, P_{d+1})$ and the tag σ_d that Anna embedded into the onion by decrypting the content C_d using the key k_d:

$$((m, H_{d+1}, k_s, K_s, P_{d+1}), \sigma_d) = \}C_d\{_{k_d}.$$

Roberto can verify that the message is untampered by running the MAC's verification algorithm $\mathsf{Ver}(K_d, M, \sigma_d)$.

To respond to the anonymous sender (Anna) with the message m', Roberto creates a new encrypted content using the keys k_s and K_s:

$$C_{d+1} = \{m', \mathsf{Tag}(K_s, m')\}_{k_s}.$$

Roberto sends the reply onion $O_{d+1} = (H_{d+1}, C_{d+1})$ to the next destination P_{d+1}. See ProcOnion and FormReply in Figures 4.4 and 4.5.

4.4.5 Processing a repliable onion (in the return direction)

Let David be an intermediary party on the return path. When David receives the onion O_j, he processes it exactly in the same way that Carol processed the onion O_i in the forward direction; he also runs ProcOnion in Figure 4.4.

4.4.6 Reading the reply

When Anna receives the onion O_s, she retrieves the reply from Roberto by first processing the onion, by running ProcOnion:

Anna first decrypts the ciphertext E_s of the header to retrieve her role "S" and the label ℓ. She reconstructs the each encryption key $k_i = F_{\mathsf{seed}}(\ell, i)$ and the authentication key K_s using the pseudo-randomness $F_{\mathsf{seed}}(s, \ell)$. (Alternatively, if she stored the keys locally, she looks up the keys associated with label ℓ in a local data structure). If O_s hasn't been tampered with, Anna retrieves the reply m' that Roberto embedded into the onion by decrypting the content C_s using the keys $(k_{d+1}, \ldots, k_s)$:

$$(m', \sigma_s) = \}\{\ldots\{C_s\}_{k_{s-1}}\ldots\}_{k_{d+1}}\{_{k_s}.$$

Anna can verify that the message is untampered by running $\mathsf{Ver}(K_s, m', \sigma_s)$. See ProcOnion in Figure 4.4.

$\mathsf{SES.FormOnion}(\ell, m, (P_1, \ldots, P_d), P^{\leftarrow})$

$(H_1, \ldots, H_d, k_1, \ldots, k_d, K_d) \leftarrow \mathsf{FormHeader}(\rightarrow, \ell, (P_1, \ldots, P_d))$
$(C_1, \ldots, C_d), H^{\leftarrow}, k^{\leftarrow}) \leftarrow \mathsf{FormContent}(\ell, m, P^{\leftarrow}, k_1, \ldots, k_d, K_d)$
return $(((H_1, C_1), \ldots, (H_d, C_d)), H^{\leftarrow}, ((k_1, \ldots, k_d, K_d), k^{\leftarrow}))$

$\mathsf{SES.FormHeader}(\mathsf{direction}, \ell, (P_1, \ldots, P_N))$

$k_j = F_{\mathsf{seed}}(\ell, j)\,, \forall j \in [N]$
$K_N = \mathsf{Gen}_{\mathsf{MAC}}(1^{\lambda}, F_{\mathsf{seed}}(N, \ell))$
$B_N^i = \}\ldots\}0\ldots0\{_{k_i}\ldots\{_{k_{N-1}}\,, \forall i \in \{1, \ldots, N-1\}$
if direction $=\rightarrow$
  $E_N \leftarrow \mathsf{Enc}(\mathsf{pk}(P_N), h(B_N^1, \ldots, B_N^{N-1}), (\mathsf{R}, k_N, K_N))$
else (if direction $=\leftarrow$)
  $E_N \leftarrow \mathsf{Enc}(\mathsf{pk}(P_N), h(B_N^1, \ldots, B_N^{N-1}), (\mathsf{S}, \ell))$
$H_N = (E_N, B_N^1, \ldots, B_N^{N-1})$
for all j from $N-1$ to 1
  $B_j^1 = \{P_N, E_N\}_{k_j}$
  $B_j^i = \{B_{j+1}^{i-1}\}_{k_j}\,, \forall i \in \{2, \ldots, N-1\}$
  $E_j \leftarrow \mathsf{Enc}(\mathsf{pk}(P_j), h(B_j^1, \ldots, B_j^{N-1}), (\mathsf{I}, k_j))$
  $H_j = (E_j, B_j^1, \ldots, B_j^{N-1})$
return $(H_1, \ldots, H_N, k_1, \ldots, k_N, K_N)$

$\mathsf{SES.FormContent}(\ell, m, (P_{d+1}, \ldots, P_s), k_1, \ldots, k_d, K_d)$

$(H_{d+1}, \ldots, H_s, k_{d+1}, \ldots, k_s, K_s) \leftarrow \mathsf{FormHeader}(\leftarrow, \ell, (P_{d+1}, \ldots, P_s))$
$M = (H_{d+1}, m, k_s, K_s, P_{d+1})$
$\sigma_d = \mathsf{Tag}(K_d, M)$
$C_d = \{M, \sigma_d\}_{k_d}$
for all j from $d-1$ to 1
  $C_j = \{C_{j+1}\}_{k_j}$
return $((C_1, \ldots, C_d), (H_{d+1}, \ldots, H_s), (k_1, \ldots, k_d, K_d))$

Figure 4.3: Pseudocode for Shallot Encryption Scheme's FormOnion. On input the label ℓ, the message m, the forward path $P^{\rightarrow}$ and the return path $P^{\leftarrow}$ (and the public keys associated with the routing path), FormOnion outputs onions $O^{\rightarrow}$, headers $H^{\leftarrow}$ and associated keys κ.

4.5 Proof that Shallot Encryption SUC-realizes $\mathcal{F}_{\mathsf{ROES}}$

In this section, we prove that our construction (in Section 4.4) SUC-realizes the ideal functionality $\mathcal{F}_{\mathsf{ROES}}$.

$\mathsf{SES.ProcOnion}(((E, B^1, \ldots, B^{N-1}), C), P)$

1 : $(\mathsf{role}, \mathsf{key}, H', P') \leftarrow \mathsf{ProcHeader}((E, B^1, \ldots, B^{N-1}), P)$
2 : $C' = \mathsf{ProcContent}(\mathsf{role}, \mathsf{key}, C)$
3 : **if** $\mathsf{role} = \mathsf{I}$, **return** $(\mathsf{I}, ((H', C'), P'))$
4 : **else**, **return** (role, C')

$\mathsf{SES.ProcHeader}((E, B^1, \ldots, B^{N-1}), P)$

1 : $t = h(B^1, \ldots, B^{N-1})$
2 : $(\mathsf{role}, \mathsf{key}) = \mathsf{Dec}(\mathsf{sk}(P), t, E)$
3 : **if** $\mathsf{role} = \mathsf{I}$
4 : $(P', E') = \}B^1\{_{\mathsf{key}}$
5 : $(B')^j = \}B^{j+1}\{_{\mathsf{key}}, \forall j \in [N-2]$
6 : $(B')^{N-1} = \}0 \ldots 0\{_{\mathsf{key}}$
7 : **return** $(\mathsf{I}, \mathsf{key}, (E', (B')^1, \ldots, (B')^{N-1}), P')$
8 : **else**
9 : **return** $(\mathsf{role}, \mathsf{key}, \perp, \perp)$

$\mathsf{SES.ProcContent}(\mathsf{role}, \mathsf{key}, C)$

1 : **if** $\mathsf{role} = \mathsf{I}$
2 : **return** $\}C\{_{\mathsf{key}}$
3 : **if** $\mathsf{role} = \mathsf{R}$
4 : parse $\mathsf{key} = (k_d, K_d)$
5 : $((H_{d+1}, m, k_s, K_s, P_{d+1}), \sigma_d) = \}C\{_{k_d}$
6 : **if** $\mathsf{Ver}(K_d, (H_{d+1}, m, k_s, K_s, P_{d+1}), \sigma_d) = 1$, **return** m
7 : **else**, **return** $\perp$
8 : **else**
9 : reconstruct keys $k_{d+1}, \ldots, k_s, K_s$ from key
10 : $(m, \sigma_s) = \}\{\ldots\{C\}_{k_{s-1}} \ldots\}_{k_{d+1}}\{_{k_s}$
11 : **if** $\mathsf{Ver}(K_s, m, \sigma_s) = 1$, **return** (ℓ, m)
12 : **else**, **return** $\perp$

Figure 4.4: Pseudocode for Shallot Encryption Scheme's ProcOnion. On input the onion $((E, B^1, \ldots, B^{N-1}), C)$ and the party P (and the secret key of P), ProcOnion returns a role (either I, R or S) and an output (either an onion and next destination $((H', C'), P')$ or a decrypted content C').

To do this, we must show that for any static setting (fixed adversary $\mathcal{A}$, set Bad of corrupted parties and public key infrastructure), there exists a simulator $\mathcal{S}$ such that for all

```
SES.FormReply(m', (H, C), P)
1 : (role, key, H', P') ← ProcHeader(H, P)
2 : if role = R
3 :    parse key = (k_d, K_d)
4 :    ((H_{d+1}, m, k_s, K_s, P_{d+1}), σ_d) = }C{_{k_d}
5 :    if Ver(K_d, (H_{d+1}, m, k_s, K_s, P_{d+1}), σ_d) = 1
6 :       σ_s = Tag(K_s, m')
7 :       return ((H_{d+1}, {m', σ_s}_{k_s}), P_{d+1})
8 :    else, return (⊥, ⊥)
9 : else, return (⊥, ⊥)
```

Figure 4.5: Pseudocode for Shallot Encryption Scheme's FormReply. On input the reply message m', the onion (H, C) and the party P (and the secret key of P), FormReply returns a return onion $(H_{d+1}, \{m', \sigma_s\}_{k_s})$ and next destination P_{d+1}.

$\mathcal{Z}$, there exists a negligible function $\nu : \mathbb{N} \mapsto \mathbb{R}$ such that

$$\left|\Pr\left[\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{ROES}},\mathcal{S},\mathcal{Z}}(1^\lambda, \mathsf{pp}) = 1\right] - \Pr\left[\mathsf{REAL}_{\mathsf{SES},\mathcal{A},\mathcal{Z}}(1^\lambda, \mathsf{pp}) = 1\right]\right| \leq \nu(\lambda),$$

where $\mathsf{REAL}_{\mathsf{SES}}$ is the environment's output in the real process.

We first provide a description of the simulator $\mathcal{S}$.

Recall that during setup, the ideal adversary (i.e., $\mathcal{S}$) sends to the ideal functionality, (i) the set $\mathcal{P}$ of participants, (ii) the set $\mathsf{Bad} \subseteq \mathcal{P}$ of corrupted parties, (iii) the onion encryption scheme's algorithms: G, ProcOnion and FormReply, (iv) the algorithm SampleOnion, (v) the algorithm CompleteOnion and (vi) the algorithm RecoverReply. (See Section 4.3.1 for the syntax of these algorithms.)

In order for our construction to be secure, the simulator $\mathcal{S}$ must provide items (i)-(vi) to $\mathcal{F}_{\mathsf{ROES}}$ such that when the ideal honest parties respond to the environment, one input at a time, the running history of outputs looks like one produced from running the real protocol using the onion encryption scheme.

To complete the description of $\mathcal{S}$, we must provide internal descriptions of how the last three items above—SampleOnion, CompleteOnion and RecoverReply—work. Since we are in the static setting, we will assume, WLOG, that these algorithms "know" who is honest, who is corrupt and all relevant keys. See Figure 4.6 for a summary of the simulator.

Send to $\mathcal{F}_{\mathsf{ROES}}$:	$\underline{\mathsf{SampleOnion}(p^{\rightarrow}, p^{\leftarrow}, m)}$
$\mathcal{P}$, Bad, G, ProcOnion, FormReply, SampleOnion, CompleteOnion, RecoverReply $\underline{\mathsf{CompleteOnion}(H', C)}$ Return the header H' along with a randomly chosen string C'	SampleOnion just runs FormOnion on the segments $p^{\rightarrow}$ and $p^{\leftarrow}$ using the all-zero label and, depending on whether the first segment contains the corrupt recipient, either the correct message m (if it does) or a random one (if it doesn't). $\underline{\mathsf{RecoverReply}(O, P)}$ Return output of ProcOnion

Figure 4.6: Summary of simulator $\mathcal{S}$

4.5.1 Description of simulator $\mathcal{S}$

Sampling an onion.

Let $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ denote the ideal functionality corresponding to the static setting. When the ideal functionality $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ receives a request from the honest party P to form an onion using the label ℓ, the message m, the forward path $P^{\rightarrow}$ and the return path $P^{\leftarrow}$, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ partitions the routing path $(P^{\rightarrow}, P^{\leftarrow})$ into "segments" where each segment is a sequence of adversarial parties that may end in a single honest party. (See Section 4.3.1 for a more formal description of these segments.) $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ runs the algorithm SampleOnion independently on each segment of the routing path. Additionally, if the forward path ends in a corrupt party, $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ runs SampleOnion on the last segment of the forward path and the first segment of the return path. Using SampleOnion in this way produces onions with the property that onions belonging to different segments are information-theoretically unrelated to each other.

The algorithm SampleOnion takes as input the security parameter 1^{λ}, the public parameters pp, the forward path $p^{\rightarrow}$ and the return path $p^{\leftarrow}$.

Case 0 The routing path $(p^{\rightarrow}, p^{\leftarrow})$ is not a sequence of adversarial parties, possibly ending in an honest party. In this case, the input is invalid, and SampleOnion returns an error.

Case 1 The return path $p^{\leftarrow}$ is non-empty and ends in an honest party P_j. In this case, SampleOnion first samples a random label $x \leftarrow\!\!\$\, \mathcal{L}(1^{\lambda})$ and then runs FormOnion on the label x, the message m (from the "form onion" request), the forward path $p^{\rightarrow} = (p_1, \ldots, p_i)$, the public keys $\mathsf{pk}(p^{\rightarrow})$ associated with the parties in $p^{\rightarrow}$, the return path $p^{\leftarrow} = (p_{i+1}, \ldots, P_j)$ and the public keys $\mathsf{pk}(p^{\leftarrow})$ associated with the parties in $p^{\leftarrow}$. Finally, SampleOnion outputs the first onion o_1 and the last header H_j in the output $((o_1, \ldots, o_i), (h_{i+1}, \ldots, H_j), \kappa) \leftarrow \mathsf{FormOnion}(1^{\lambda}, \mathsf{pp}, x, m, p^{\rightarrow}, \mathsf{pk}(p^{\rightarrow}), p^{\leftarrow}, \mathsf{pk}(p^{\leftarrow}))$.

Case 2 The return path $p^{\leftarrow}$ is empty, and the forward path $p^{\rightarrow}$ ends in an honest party P_i. In this case, $\mathsf{SampleOnion}$ first samples a random label $x \leftarrow\!\!\$\, \mathcal{L}(1^\lambda)$ and a random message $y \leftarrow\!\!\$\, \mathcal{M}(1^\lambda)$ and then runs $\mathsf{FormOnion}$ on the label x, the message y, the forward path $p^{\rightarrow} = (p_1, \ldots, P_i)$, the public keys $\mathsf{pk}(p^{\rightarrow})$ associated with the parties in $p^{\rightarrow}$, the empty return path "()" and the empty sequence "()" of public keys. Finally, $\mathsf{SampleOnion}$ outputs the first onion o_1 and the last onion O_i in the output $((o_1, \ldots, O_i), (), \kappa) \leftarrow \mathsf{FormOnion}(1^\lambda, \mathsf{pp}, x, y, p^{\rightarrow}, \mathsf{pk}(p^{\rightarrow}), (), ())$.

Case 3 The return path $p^{\leftarrow}$ is empty, and the forward path $p^{\rightarrow}$ ends in a corrupt party p_i. In this case, $\mathsf{SampleOnion}$ first samples a random label $x \leftarrow\!\!\$\, \mathcal{L}(1^\lambda)$ and then runs $\mathsf{FormOnion}$ on the label x, the message m (from the "form onion" request), the forward path $p^{\rightarrow} = (p_1, \ldots, p_i)$, the public keys $\mathsf{pk}(p^{\rightarrow})$ associated with the parties in $p^{\rightarrow}$, the empty return path "()" and the empty sequence "()" of public keys. Finally, $\mathsf{SampleOnion}$ outputs the first onion o_1 in the output $((o_1, \ldots, o_i), h^{\leftarrow}, \kappa) \leftarrow \mathsf{FormOnion}(1^\lambda, \mathsf{pp}, x, m, p^{\rightarrow}, \mathsf{pk}(p^{\rightarrow}), (), ())$.

Completing an onion.

The environment $\mathcal{Z}$ can modify just the content of an honestly formed onion $O = (H, X)$, leaving the header H intact. When $\mathcal{Z}$ instructs an honest party P to process this kind of onion $O = (H, C)$, the ideal functionality $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ runs the algorithm $\mathsf{CompleteOnion}$ to produce an onion (H', C') that (i) looks like the output of $\mathsf{ProcOnion}$ on (H, C) and (ii) has the same header H' that $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ assigned to the peeled onion (H', X') of (H, X).

To do this, the algorithm $\mathsf{CompleteOnion}(1^\lambda, \mathsf{pp}, H', C)$ samples a random string $C' \leftarrow\!\!\$\, \{0,1\}^{L_2(\lambda)}$, where $\{0,1\}^{L_2(\lambda)}$ corresponds to the blocklength for the PRP (in the construction), and outputs (H', C').

Recovering a reply message.

The environment $\mathcal{Z}$ can instruct an honest party P to process a return onion O formed by a corrupt recipient p_d in response to an onion from an honest sender; P can be an intermediary party on the return path or the original sender. In such a situation, the ideal functionality $\mathcal{F}_{\mathsf{ROES}}^{\mathsf{sid}}$ runs the algorithm $\mathsf{RecoverReply}$ to recover the reply message from O.

The algorithm $\mathsf{RecoverReply}(1^\lambda, \mathsf{pp}, O, P)$ simply runs $\mathsf{ProcOnion}(O, P, \mathsf{sk}(P))$ and returns the message in the output.

4.5.2 A cryptographic definition of security

To prove that our onion encryption scheme SUC-realizes the ideal functionality $\mathcal{F}_{\mathsf{ROES}}$, it will be useful to first show that the scheme satisfies *repliable-onion security*. Informally, an onion encryption scheme is repliable-onion secure if the adversary cannot tell (i) whether an honest receiver of an honestly formed onion is an intermediary for the onion or the recipient, (ii) whether an honest transmitter of an honestly formed onion is an intermediary for the onion or the sender and (iii) how far an honestly formed onion is from its origin and destination.

Formally, we define repliable-onion security using the game, $\mathsf{ROSecurityGame}$. See Figure 4.7 for a summary of the game.

1: $\mathcal{A}$ picks honest parties' router names Q_{I} and Q_{S}

2: $\mathcal{C}$ sets keys for honest parties

3: $\mathcal{A}$ gets oracle access to oracles—$\mathcal{O}.\mathsf{PO}_{\mathsf{I}}$, $\mathcal{O}.\mathsf{FR}_{\mathsf{I}}$, $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ and $\mathcal{O}.\mathsf{FR}_{\mathsf{S}}$— for processing onions and replying to them on behalf of Q_{I} and Q_{S}

4: $\mathcal{A}$ provides input for challenge onion

5: $\mathcal{C}$ flips a coin $b \leftarrow_{\$} \{0,1\}$

6: If $b = 0$, $\mathcal{C}$ forms onion specified by $\mathcal{A}$

7: If $b = 1$, $\mathcal{C}$ forms onion with "switch" at Q_{I} and modifies oracles accordingly.

   (a) If Q_{I} is on the forward path, to peel the onion on behalf of Q_{I}, $\mathcal{O}.\mathsf{PO}_{\mathsf{I}}$ forms a new onion using the remainder of the routing path

   (b) If Q_{I} is the recipient, to form a reply on behalf of Q_{I}, $\mathcal{O}.\mathsf{FR}_{\mathsf{I}}$ forms a new onion using the return path as the forward path (and the empty return path)

   (c) If Q_{I} is on the return path, to peel the onion of behalf of Q_{I}, $\mathcal{O}.\mathsf{PO}_{\mathsf{I}}$ forms a new onion using the remainder of the return path as the forward path (and the empty return path)

8: $\mathcal{A}$ gets oracle access to $\mathcal{O}.\mathsf{PO}_{\mathsf{I}}$, $\mathcal{O}.\mathsf{FR}_{\mathsf{I}}$, $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ and $\mathcal{O}.\mathsf{FR}_{\mathsf{S}}$

9: $\mathcal{A}$ guesses b' and wins if $b' = b$

Figure 4.7: Summary of the repliable onion security game, $\mathsf{ROSecurityGame}$.

$\mathsf{ROSecurityGame}(1^{\lambda}, \Sigma, \mathsf{CompleteOnion}, \mathcal{A})$ is parametrized by the security parameter 1^{λ}, the repliable onion encryption scheme $\Sigma = (G, \mathsf{FormOnion}, \mathsf{ProcOnion}, \mathsf{FormReply})$, the p.p.t. algorithm $\mathsf{CompleteOnion}$ and the adversary $\mathcal{A}$.

1. The adversary $\mathcal{A}$ picks two router names $Q_{\mathsf{I}}, Q_{\mathsf{S}} \in \mathcal{P}$ ("I", for intermediary and "S,

for sender) and sends them to the challenger $\mathcal{C}$.

2. The challenger $\mathcal{C}$ generates key pairs $(\mathsf{pk}(Q_\mathsf{I}), \mathsf{sk}(Q_\mathsf{I}))$ and $(\mathsf{pk}(Q_\mathsf{S}), \mathsf{sk}(Q_\mathsf{S}))$ for Q_I and Q_S using the key generation algorithm G and sends the public keys $(\mathsf{pk}(Q_\mathsf{I}), \mathsf{pk}(Q_\mathsf{S}))$ to $\mathcal{A}$.

3. $\mathcal{A}$ is given oracle access to (i) $\mathcal{O}.\mathsf{PO}_\mathsf{I}(\cdot)$, (ii) $\mathcal{O}.\mathsf{FR}_\mathsf{I}(\cdot,\cdot)$, (iii) $\mathcal{O}.\mathsf{PO}_\mathsf{S}(\cdot)$ and (iv) $\mathcal{O}.\mathsf{FR}_\mathsf{S}(\cdot,\cdot)$ where

   i-ii. $\mathcal{O}.\mathsf{PO}_\mathsf{I}(\cdot)$ and $\mathcal{O}.\mathsf{FR}_\mathsf{I}(\cdot,\cdot)$ are, respectively, the oracle for answering "process onion" requests made to honest party Q_I and the oracle for answering "form reply" requests made to Q_I.

   iii-iv. $\mathcal{O}.\mathsf{PO}_\mathsf{S}(\cdot)$ and $\mathcal{O}.\mathsf{FR}_\mathsf{S}(\cdot,\cdot)$ are, respectively, the oracle for answering "process onion" requests made to honest party Q_S and the oracle for answering "form reply" requests made to Q_S, i.e.,

$$\begin{aligned}
\mathcal{O}.\mathsf{PO}_\mathsf{I}(O) &= \mathsf{ProcOnion}(O, Q_\mathsf{I}, \mathsf{sk}(Q_\mathsf{I})) \\
\mathcal{O}.\mathsf{FR}_\mathsf{I}(m', O) &= \mathsf{FormReply}(m', O, Q_\mathsf{I}, \mathsf{sk}(Q_\mathsf{I})) \\
\mathcal{O}.\mathsf{PO}_\mathsf{S}(O) &= \mathsf{ProcOnion}(O, Q_\mathsf{S}, \mathsf{sk}(Q_\mathsf{S})) \\
\mathcal{O}.\mathsf{FR}_\mathsf{S}(m', O) &= \mathsf{FormReply}(m', O, Q_\mathsf{S}, \mathsf{sk}(Q_\mathsf{S}))
\end{aligned}$$

   Since $\mathsf{ProcOnion}$ and $\mathsf{FormReply}$ are deterministic algorithms, WLOG, the oracles don't respond to repeating queries.

4. $\mathcal{A}$ chooses a label $\ell \in \mathcal{L}(1^\lambda)$ and a message $m \in \mathcal{M}(1^\lambda)$. $\mathcal{A}$ also chooses a forward path $P^{\rightarrow} = (P_1, \ldots, P_d)$ and a return path $P^{\leftarrow} = (P_{d+1}, \ldots, P_s)$ such that exactly one party P_j in the routing path $(P^{\rightarrow}, P^{\leftarrow})$ is equal to Q_I, and only the last party P_s is equal to Q_S. $\mathcal{A}$ sends to $\mathcal{C}$ the parameters for the challenge onion: ℓ, m, $P^{\rightarrow}$, the public keys $\mathsf{pk}(P^{\rightarrow})$ of the parties in $P^{\rightarrow}$, $P^{\leftarrow}$ and the public keys $\mathsf{pk}(P^{\leftarrow})$ of the parties in $P^{\leftarrow}$.

5. $\mathcal{C}$ samples a bit $b \leftarrow\!\!\$\, \{0,1\}$.

   If $b = 0$, $\mathcal{C}$ runs $\mathsf{FormOnion}$ on the parameters specified by $\mathcal{A}$, i.e.,

$$((O_1^0, \ldots, O_d^0), H^{\leftarrow}, \kappa) \leftarrow \mathsf{FormOnion}(\ell, m, P^{\rightarrow}, \mathsf{pk}(P^{\rightarrow}), P^{\leftarrow}, \mathsf{pk}(P^{\leftarrow})).$$

   The oracles—$\mathcal{O}.\mathsf{PO}_\mathsf{I}(\cdot)$, $\mathcal{O}.\mathsf{FR}_\mathsf{I}(\cdot,\cdot)$, $\mathcal{O}.\mathsf{PO}_\mathsf{S}(\cdot)$ and $\mathcal{O}.\mathsf{FR}_\mathsf{S}(\cdot,\cdot)$—remain unmodified.

   Otherwise, if $b = 1$,

(a) If $j < d$, $\mathcal{C}$ performs the "switch" at honest party $P_j = Q_{\mathsf{I}}$ on the forward path $P^{\rightarrow}$. $\mathcal{C}$ runs $\mathsf{FormOnion}$ twice. First, $\mathcal{C}$ runs it on input a random label $x \leftarrow\!\!\$\, \mathcal{L}(1^\lambda)$, a random message $y \leftarrow\!\!\$\, \mathcal{M}(1^\lambda)$, the "truncated" forward path $p^{\rightarrow} = (P_1, \ldots, P_j)$ and the empty return path "()", i.e.,

$$((O_1^1, \ldots, O_j^1), (), \kappa) \leftarrow \mathsf{FormOnion}(x, y, p^{\rightarrow}, \mathsf{pk}(p^{\rightarrow}), (), ()).$$

$\mathcal{C}$ then runs $\mathsf{FormOnion}$ on a random label $x' \leftarrow\!\!\$\, \mathcal{L}(1^\lambda)$, the message m (that had been chosen by $\mathcal{A}$ in step 4), the remainder $q^{\rightarrow} = (P_{j+1}, \ldots, P_d)$ of the forward path and the return path $P^{\leftarrow}$, i.e.,

$$((O_{j+1}^1, \ldots, O_d^1), H^{\leftarrow}, \kappa') \leftarrow \mathsf{FormOnion}(x', m, q^{\rightarrow}, \mathsf{pk}(q^{\rightarrow}), P^{\leftarrow}, \mathsf{pk}(P^{\leftarrow})),$$

We modify the oracles as follows. Let $O_j^1 = (H_j^1, C_j^1)$ and $O_{j+1}^1 = (H_{j+1}^1, C_{j+1}^1)$, and let H_s^1 be the last header in $H^{\leftarrow}$. $\mathcal{O}.\mathsf{PO}_{\mathsf{I}}$ does the following to "process" an onion $O = (H, C)$:

i. If $O = O_j^1$ and $\mathsf{ProcOnion}(O, P_j, \mathsf{sk}(P_j)) = (\mathsf{R}, y)$, then return $(\mathsf{I}, (O_{j+1}^1, P_{j+1}))$.

ii. If $O = O_j^1$ and $\mathsf{ProcOnion}(O, P_j, \mathsf{sk}(P_j)) \neq (\mathsf{R}, y)$, then fail.

iii. If $O \neq O_j^1$ but $H = H_j^1$ and $\mathsf{ProcOnion}(O, P_j, \mathsf{sk}(P_j)) = (\mathsf{R}, \perp)$, then return $(\mathsf{I}, ((H_{j+1}^1, \mathsf{CompleteOnion}(H_{j+1}^1, C)), P_{j+1}))$.

iv. If $O \neq O_j^1$ but $H = H_j^1$ and $\mathsf{ProcOnion}(O, P_j, \mathsf{sk}(P_j)) \neq (\mathsf{R}, \perp)$, then fail.

$\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ does the following to "process" an onion O:

v. If the header of O is H_s^1 and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) = (\mathsf{R}, m')$ for some message $m' \neq \perp$, then return $(\mathsf{S}, (\ell, m'))$.

vi. If the header of O is H_s^1 and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) = (\mathsf{R}, \perp)$, then return $(\mathsf{S}, \perp)$.

vii. If the header of O is H_s^1 and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) \neq (\mathsf{R}, m')$ for any message m', then fail.

All other queries are processed as before.

(b) If $j = d$, $\mathcal{C}$ performs the "switch" at honest recipient P_j. $\mathcal{C}$ runs $\mathsf{FormOnion}$ on input a random label $x \leftarrow\!\!\$\, \mathcal{L}(1^\lambda)$, a random message $y \leftarrow\!\!\$\, \mathcal{M}(1^\lambda)$, the forward path $P^{\rightarrow}$ and the empty return path "()", i.e.,

$$((O_1^1, \ldots, O_j^1), (), \kappa) \leftarrow \mathsf{FormOnion}(x, y, P^{\rightarrow}, \mathsf{pk}(P^{\rightarrow}), (), ()).$$

We modify the oracles as follows. $\mathcal{O}.\mathsf{FR}_{\mathsf{I}}$ does the following to "form a reply" using message m' and onion $O = O_j^1$: $\mathcal{O}.\mathsf{FR}_{\mathsf{I}}$ runs $\mathsf{FormOnion}$ on a random label x', the reply message m', the return path $P^{\leftarrow}$ as the forward path and the empty return path "()", i.e.,

$$((O_{j+1}^{m'}, \ldots, O_s^{m'}), (), \kappa^{m'}) \leftarrow \mathsf{FormOnion}(x', m', P^{\leftarrow}, \mathsf{pk}(P^{\leftarrow}), (), ()),$$

stores the pair $(O_s^{m'}, m')$ (such that the pair is accessible by $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$) and returns $(O_{j+1}^{m'}, P_{j+1})$.

$\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ does the following to "process" an onion O:

i. If $O = O'$ for some stored pair (O', m') and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) = (\mathsf{R}, m')$, then return $(\mathsf{S}, (\ell, m'))$.

ii. If $O = O'$ for some stored pair (O', m') and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) \neq (\mathsf{R}, m')$, then fail.

iii. If $O \neq O'$ for any stored pair (O', m') but $O = (H', C)$ for some stored pair $((H', C'), m')$ and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) = (\mathsf{R}, \perp)$, then return $(\mathsf{S}, \perp)$.

iv. If $O \neq O'$ for any stored pair (O', m') but $O = (H', C)$ for some stored pair $((H', C'), m')$ and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) \neq (\mathsf{R}, \perp)$, then fail.

All other queries are processed as before.

(c) If $j > d$, $\mathcal{C}$ performs the "switch" at honest party P_j on the return path $P^{\leftarrow}$. $\mathcal{C}$ runs $\mathsf{FormOnion}$ on input a random label $x \leftarrow\!\!\$\, \mathcal{L}(1^\lambda)$, the message m (that had been chosen by $\mathcal{A}$ in step 4), the forward path $P^{\rightarrow}$ and the "truncated" return path $p^{\leftarrow} = (P_{d+1}, \ldots, P_j)$, i.e.,

$$(O^{\rightarrow}, (H_{d+1}^1, \ldots, H_j^1), \kappa) \leftarrow \mathsf{FormOnion}(x, m, P^{\rightarrow}, \mathsf{pk}(P^{\rightarrow}), p^{\leftarrow}, \mathsf{pk}(p^{\leftarrow})).$$

We modify the oracles as follows. $\mathcal{O}.\mathsf{PO}_{\mathsf{I}}$ does the following to "process" an onion O:

i. If $O = (H_j^1, C)$ for some content C and $\mathsf{ProcOnion}(O, P_j, \mathsf{sk}(P_j)) = (\mathsf{R}, m')$ for some message m' (possibly equal to "$\perp$"), then runs $\mathsf{FormOnion}$ on a random label x', the reply message m', the remainder the return path $q^{\leftarrow} = (P_{j+1}, \ldots, P_s)$ as the forward path and the empty return path "()", i.e.,

$$((O_{j+1}^{m'}, \ldots, O_s^{m'}), (), \kappa^{m'}) \leftarrow \mathsf{FormOnion}(x', m', q^{\leftarrow}, \mathsf{pk}(q^{\leftarrow}), (), ()),$$

stores the pair $(O_s^{m'}, m')$ (such that the pair is accessible by $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$) and returns $(O_{j+1}^{m'}, P_{j+1})$.

ii. If $O = (H_j^1, C)$ for some content C and $\mathsf{ProcOnion}(O, P_j, \mathsf{sk}(P_j)) \neq (\mathsf{R}, m')$ for some message m', then fails.

$\mathcal{O}.\mathsf{PO}_\mathsf{S}$ does the following to "process" an onion O:

iii. If $O = O'$ for some stored pair (O', m') and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) = (\mathsf{R}, m')$, then return $(\mathsf{S}, (\ell, m'))$.

iv. If $O = O'$ for some stored pair (O', m') and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) \neq (\mathsf{R}, m')$, then fail.

v. If $O \neq O'$ for any stored pair (O', m') but $O = (H', C)$ for some stored pair $((H', C'), m')$ and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) = (\mathsf{R}, \perp)$, then return $(\mathsf{S}, \perp)$.

vi. If $O \neq O'$ for any stored pair (O', m') but $O = (H', C)$ for some stored pair $((H', C'), m')$ and $\mathsf{ProcOnion}(O, P_s, \mathsf{sk}(P_s)) \neq (\mathsf{R}, \perp)$, then fail.

All other queries are processed as before.

$\mathcal{C}$ sends to $\mathcal{A}$, the first onion O_1^b in the output of $\mathsf{FormOnion}$.

6. $\mathcal{A}$ submits a polynomially-bounded number of (adaptively chosen) queries to oracles $\mathcal{O}.\mathsf{PO}_\mathsf{I}(\cdot)$, $\mathcal{O}.\mathsf{FR}_\mathsf{I}(\cdot,\cdot)$, $\mathcal{O}.\mathsf{PO}_\mathsf{S}(\cdot)$ and $\mathcal{O}.\mathsf{FR}_\mathsf{S}(\cdot,\cdot)$.

7. Finally, $\mathcal{A}$ guesses a bit b' and wins if $b' = b$.

We define *repliable-onion security* as follows.

Definition 16 (Repliable-onion security)**.** *A repliable onion encryption scheme* Σ *is* repliable-onion secure *if there exist a p.p.t. algorithm* $\mathsf{CompleteOnion}$ *and a negligible function* ν *such that every p.p.t. adversary* $\mathcal{A}$ *wins the security game* $\mathsf{ROSecurityGame}(1^\lambda, \Sigma, \mathsf{CompleteOnion}, \mathcal{A})$ *with negligible advantage, i.e.,*

$$\left|\Pr\left[\mathcal{A} \textit{ wins } \mathsf{ROSecurityGame}(1^\lambda, \Sigma, \mathsf{CompleteOnion}, \mathcal{A})\right] - \frac{1}{2}\right| \leq \nu(\lambda).$$

Remarks on Defintion 16 An onion formed by running a secure onion encryption scheme and received (resp. transmitted) by an honest party P does not reveal how many layers are remaining (resp. came before) since the adversary cannot distinguish between the onion and another onion formed using the same parameters except with the path truncating at recipient (resp. sender) P.

In the security game, the adversary chooses a routing path with only one honest party (besides the necessarily honest sender). Restricting the adversary's choice for a routing

path in this way simplifies the definition of security without restricting the usefulness of the definition since each segment of an onion formed using a secure scheme (the way we defined it) must be computationally unrelated to any other segment.

4.5.3 Security results

We first argue that our construction satisfies repliable-onion security.

Lemma 12. *Shallot Encryption Scheme (in Section 4.4) is correct (Definition 1) and repliable-onion secure (Definition 16) under the assumption that (i) $\{f_k\}_{k\in\{0,1\}^*}$ is a PRP, (ii) $\mathcal{E}$ is a CCA2-secure encryption scheme with tags, (iii)* MAC *is a message authentication code, and (iv) h is a collision-resistant hash function.*

Proof idea: We sketch the proof of security for cases (a) when $P_j = Q_1$ is an intermediary on the forward path and (c) when P_j is an intermediary on the return path. The proof for case (b) (when P_j is the recipient) is similar.

In cases (a) and (c), we can prove that $\mathcal{A}$'s view when $b = 0$ is indistinguishable from $\mathcal{A}$'s view when $b = 1$ using a hybrid argument. The gist of the argument is as follows: First, P_j's encryption key k_j is protected by CCA-secure encryption, so it can be swapped out for the all-zero key "$0 \ldots 0$". Next, blocks $(N - j - 1)$ to $(N - 1)$ of the onion for P_{j+1} look random as they are all "decryptions" under k_j, so they can be swapped out for truly random blocks. Next, blocks 1 to $(N - j - 1)$ and the content of the onion for P_j look random as they are encryptions under k_j, so they can be swapped out for truly random blocks. At this point, the keys for forming O_{j+1} can be independent of the keys for forming O_j, and these onions may be formed via separate FormOnion calls. For case (b), we can use a simpler hybrid argument since only the content of a forward onion can be computationally related to the keys for the return path. Thus, we can swap out just the content for a truly random string.

For the full proof, see below.

Proof. The onion encryption scheme is correct by inspection.

We present the proof of security for case (a) when the switch occurs at intermediary $P_j = Q_{\mathsf{I}}$ on the forward path. The proofs for cases (b) and (c) are similar.

For the analysis of the scheme's repliable-onion security, we will make the simplifying assumption that labels are truly random as opposed to generated using a PRF. We can make this assumption "without loss in rigor" since a proof that relies on this assumption implies one without making the assumption.

To prove the lemma, we need to prove that $\mathcal{A}$ cannot distinguish between running $\mathsf{Experiment}^0$ (game with $b = 0$) and $\mathsf{Experiment}^1$ (game with $b = 1$). To do this, we define hybrids Hybrid^1 through Hybrid^9 and prove that (i) running Hybrid^1 produces the same result that running $\mathsf{Experiment}^0$ produces, (ii) $\mathcal{A}$ cannot distinguish between running any two consecutive hybrids, and (iii) running Hybrid^9 produces the same result that running $\mathsf{Experiment}^1$ produces. See Figure 4.8 for the road map of the proof.

$\mathsf{Experiment}^0$—game with $b = 0$, same as Hybrid^1
Hybrid^1—make H_{d+1}, then O_{j+1}, then O_1
Hybrid^2—same as Hybrid^1 except swap ℓ for random label
Hybrid^3—same as Hybrid^2 except swap k_j for fake key "$0 \ldots 0$"
Hybrid^4—same as Hybrid^3 except swap $(B_{j+1}^{N-j-1}, \ldots, B_{j+1}^{N-1})$ for truly random blocks
Hybrid^5—same as Hybrid^4 except swap $(B_j^1, \ldots, B_j^{N-j-1})$ and content C_j for truly random strings
Hybrid^6—same as Hybrid^5 except swap onion for intermediary P_j for onion for recipient P_j
Hybrid^7—same as Hybrid^6 except swap truly random blocks and content in O_j for pseudo-random blocks $(B_j^1, \ldots, B_j^{N-j-1}, C_j)$
Hybrid^8—same as Hybrid^7 except swap truly random blocks in H_{j+1} for pseudo-random blocks $(B_{j+1}^{N-j-1}, \ldots, B_{j+1}^{N-1})$
Hybrid^9—same as Hybrid^8 except swap key "$0 \ldots 0$" for for real key k_j
$\mathsf{Experiment}^1$—game with $b = 1$, same as Hybrid^9

Figure 4.8: Road map of proof of Lemma 12

Security game with $b = 1$ Let $\mathsf{Experiment}^1$ be the challenger's algorithm in the security game when $b = 1$. In $\mathsf{Experiment}^1$, the challenger does the following:

1: get from $\mathcal{A}$ router names Q_{I} and Q_{S} and sends $(Q_{\mathsf{I}}, Q_{\mathsf{S}})$

2: generate keys for Q_{I} and Q_{S} and sends public keys $(\mathsf{pk}(Q_{\mathsf{I}}), \mathsf{pk}(Q_{\mathsf{S}}))$ to $\mathcal{A}$

3: give $\mathcal{A}$ oracle access to $\mathcal{O}.\mathsf{PO}_{\mathsf{I}}$, $\mathcal{O}.\mathsf{FR}_{\mathsf{I}}$, $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ and $\mathcal{O}.\mathsf{FR}_{\mathsf{S}}$

4: get from $\mathcal{A}$ parameters for challenge onion: label ℓ, message m, forward path $P^{\rightarrow} = (P_1, \ldots, P_d)$ and return path $P^{\leftarrow} = (P_{j+1}, \ldots, P_s)$ such that $P_j = Q_{\mathsf{I}}$ and $P_s = Q_{\mathsf{S}}$, and the public keys of the adversarial parties in the routing path

5: pick a random message $y \leftarrow\!\!\$\, \mathcal{M}(1^\lambda)$;
let $p^{\rightarrow} = (P_1, \ldots, P_j)$ and $q^{\rightarrow} = (P_{j+1}, \ldots, P_d)$;
run $\mathsf{FormOnion}((0 \ldots 0), y, p^{\leftarrow}, \mathsf{pk}(p^{\leftarrow}), (), ()) \rightarrow ((O_1, \ldots, O_j), (), \kappa)$;
run $\mathsf{FormOnion}((0 \ldots 0), m, q^{\leftarrow}, \mathsf{pk}(q^{\leftarrow}), P^{\leftarrow}, \mathsf{pk}(P^{\leftarrow})) \rightarrow ((O_{j+1}, \ldots, O_d), H^{\leftarrow}, \kappa')$

6: modify $\mathcal{O}.\mathsf{PO}_\mathsf{I}$: to "process" an onion $O = (H_j, C)$ with the same header H_j as O_j, $\mathcal{O}.\mathsf{PO}_\mathsf{I}$ returns $(\mathsf{I}, (\mathsf{CompleteOnion}(1^\lambda, \mathsf{pp}, H_{j+1}, C), P_{j+1})$;
modify $\mathcal{O}.\mathsf{PO}_\mathsf{S}$: to "process" an onion O with the header H_s (i.e., last header in $H^{\leftarrow}$), $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ extracts the message $m' = \mathsf{RecoverReply}(\lambda, \mathsf{pp}, O, P_j, \mathsf{sk}(P_j))$ and returns $(\mathsf{S}, (\ell, m'))$

7: send to $\mathcal{A}$ the first onion O_1

8: give $\mathcal{A}$ oracle access to $\mathcal{O}.\mathsf{PO}_\mathsf{I}$, $\mathcal{O}.\mathsf{FR}_\mathsf{I}$, $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ and $\mathcal{O}.\mathsf{FR}_\mathsf{S}$

Security game with $b = 0$ Let $\mathsf{Experiment}^0$ be the challenger's algorithm in the security game when $b = 0$. In $\mathsf{Experiment}^0$, the challenger does the following:

1: get from $\mathcal{A}$ router names Q_I and Q_S and sends $(Q_\mathsf{I}, Q_\mathsf{S})$

2: generate keys for Q_I and Q_S and sends public keys $(\mathsf{pk}(Q_\mathsf{I}), \mathsf{pk}(Q_\mathsf{S}))$ to $\mathcal{A}$

3: give $\mathcal{A}$ oracle access to $\mathcal{O}.\mathsf{PO}_\mathsf{I}$, $\mathcal{O}.\mathsf{FR}_\mathsf{I}$, $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ and $\mathcal{O}.\mathsf{FR}_\mathsf{S}$

4: get from $\mathcal{A}$ parameters for challenge onion: label ℓ, message m, forward path $P^{\rightarrow} = (P_1, \ldots, P_d)$ and return path $P^{\leftarrow} = (P_{j+1}, \ldots, P_s)$ such that $P_j = Q_\mathsf{I}$ and $P_s = Q_\mathsf{S}$, and the public keys of the adversarial parties in the routing path

5: run $\mathsf{FormOnion}(\ell, m, P^{\rightarrow}, \mathsf{pk}(P^{\rightarrow}), P^{\leftarrow}, \mathsf{pk}(P^{\leftarrow})) \rightarrow ((O_1, \ldots, O_d), H^{\leftarrow}, \kappa)$

6: keep the oracles unmodified

7: send to $\mathcal{A}$ the first onion O_1

8: give $\mathcal{A}$ oracle access to $\mathcal{O}.\mathsf{PO}_\mathsf{I}$, $\mathcal{O}.\mathsf{FR}_\mathsf{I}$, $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ and $\mathcal{O}.\mathsf{FR}_\mathsf{S}$

Hybrid^1—**make H_{d+1}, then O_{j+1}, then O_1** Let Hybrid^1 be the same procedure as $\mathsf{Experiment}^0$ except for step 5.

In step 5, rather than using $\mathsf{FormOnion}$ as a black box to obtain O_1, the challenger forms onion O_{j+1} by running $\mathsf{FormHeader}(\rightarrow, \ell, (P_{j+1}, \ldots, P_d))$ (to get the header H_{j+1}) and $\mathsf{FormContent}(\ell, m, P^{\leftarrow}, k_{j+1}, \ldots, k_d, K_d)$ (to get the content C_{j+1}); and finally produces onion O_1 by wrapping onion layers around $O_{j+1} = (H_{j+1}, C_{j+1})$:

5: form onion O_{j+1}:

$$(H_{j+1}, \ldots, H_d, \kappa) \leftarrow \mathsf{FormHeader}(\rightarrow, \ell, (P_{j+1}, \ldots, P_d))$$
$$((C_{j+1}, \ldots, C_d), H^{\leftarrow}, \kappa') \leftarrow \mathsf{FormContent}(\ell, m, P^{\leftarrow}, \kappa);$$

form onion O_1 by wrapping layers around $O_{j+1} = ((E_{j+1}, B^1_{j+1}, \ldots, B^{N-1}_{j+1}), C_{j+1})$ using the keys in κ; for all u from j to one, recursively obtain O_u from O_{u+1} as follows:

$$\begin{aligned} B^1_u &= \{P_{u+1}, E_{u+1}\}_{k_u} \\ \forall i \in \{2, \ldots, N-1\},\ B^i_u &= \{B^{i-1}_{u+1}\}_{k_u} \\ E_u &\leftarrow \mathsf{Enc}(\mathsf{pk}(P_u), h(B^1_u, \ldots, B^{N-1}_u), k_u) \\ C_u &= \{C_{u+1}\}_{k_u} \end{aligned}$$

6: keep the oracles unmodified

Hybrid^1 is the same procedure as $\mathsf{Experiment}^0$ "under the hood".

Hybrid^2—swap ℓ for random label Let Hybrid^2 be the same procedure as Hybrid^1 except in steps 5-6, the challenger swaps out the label ℓ (from $\mathcal{A}$) for a random label x and modifies oracle $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ accordingly:

5: form onion O_{j+1}:

$$x \leftarrow \mathcal{L}(1^\lambda)$$
$$(H_{j+1}, \ldots, H_d, \kappa) \leftarrow \mathsf{FormHeader}(\rightarrow, x, (P_{j+1}, \ldots, P_d))$$
$$((C_{j+1}, \ldots, C_d), H^{\leftarrow}, \kappa') \leftarrow \mathsf{FormContent}(x, m, P^{\leftarrow}, \kappa);$$

form onion O_1 by wrapping layers around O_{j+1} (using keys produced from forming O_{j+1})

6: modify $\mathcal{O}.\mathsf{PO}_\mathsf{S}$: to "process" an onion O with the header H_s (i.e., last header in $H^{\leftarrow}$), $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ extracts the message $m' = \mathsf{RecoverReply}(\lambda, \mathsf{pp}, O, P_j, \mathsf{sk}(P_j))$ and returns $(\mathsf{S}, (\ell, m'))$

Here, we prove that $\mathcal{A}$ cannot distinguish between running Hybrid^1 and running Hybrid^2. For the sake of reaching a contradiction, suppose that $\mathcal{A}$ can distinguish between running Hybrid^1 (i.e., $b = 0$) and running Hybrid^2 (i.e., $b = 1$), then we can construct a reduction $\mathcal{B}$ that can break the CCA2-security of the underlying encryption scheme as follows:

1: $\mathcal{B}$ receives the router names $Q_{\mathsf{I}}, Q_{\mathsf{S}}$ from $\mathcal{A}$.

2: $\mathcal{B}$ generates keys $(\mathsf{pk}(Q_{\mathsf{I}}), \mathsf{sk}(Q_{\mathsf{I}}))$ for Q_{I} using the key generation algorithm G but gets the public key $\mathsf{pk}(Q_{\mathsf{S}})$ of Q_{S} from its challenger. $\mathcal{B}$ sends the public keys $(\mathsf{pk}(Q_{\mathsf{I}}), \mathsf{pk}(Q_{\mathsf{S}}))$ to $\mathcal{A}$.

3: $\mathcal{B}$ gives oracle access to $\mathcal{A}$; whenever $\mathcal{B}$ needs to process an onion $O = ((E, B^1, \ldots, B^{N-1}), C)$ for Q_{S}, $\mathcal{B}$ uses the decryption oracle $\mathcal{O}.\mathsf{Dec}$ to decrypt the ciphertext portion E of O. For all other "process onion" requests, $\mathcal{B}$ simply runs $\mathsf{ProcOnion}$.

4: $\mathcal{B}$ gets from $\mathcal{A}$ the challenge onion parameters: label ℓ, message m, forward path $P^{\rightarrow}$ and return path $P^{\leftarrow}$ and the public keys of the adversarial parties in the routing path. $\mathcal{B}$ sends the challenge messages $m^0 = \ell$ and $m^1 \leftarrow\!\!\$\, \mathcal{L}(1^\lambda)$ to the challenger, and the challenger responds with the encryption E_s^b of one of the messages.

5: Let $|k|(\lambda)$ be the length of the encryption keys. $\mathcal{B}$ uses E_s^b to form header H'_s:

$$
\begin{aligned}
k_1, \ldots, k_N &\leftarrow\!\!\$\, \{0,1\}^{|k|(\lambda)} \\
B_s^1 &= \{\bot, \bot\}_{k_N} \\
\forall i \{2, \ldots, N-1\},\ B_s^i &= \}\ldots\}0\ldots0\{_{k_i}\ldots\{_{k_{N-1}} \\
H'_s &= (E_s^b, B_s^1, \ldots, B_s^{N-1})
\end{aligned}
$$

and forms header H'_{d+1} by "wrapping" H'_s (using the return path $P^{\leftarrow}$ and the keys $k_1, \ldots, k_{N-1}$). $\mathcal{B}$ then forms onion O_{j+1} by running $\mathsf{FormHeader}(\rightarrow, x, (P_{j+1}, \ldots, P_d))$ and $\mathsf{FormContent}(x, m, P^{\leftarrow}, k_{j+1}, \ldots, k_d, K_d)$ but replacing the internally created H_{d+1} with H'_{d+1}. Finally, $\mathcal{B}$ forms onion O_1 by wrapping onion layers around O_{j+1}.

6: $\mathcal{B}$ (possibly) modifies $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ so that if running $\mathsf{ProcOnion}$ on an onion with header H_s outputs $(\mathsf{S}, (x, m'))$ for some label x and message m, $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ outputs $(\mathsf{S}, (\ell, m'))$ instead (all other "process onion" requests are handled by running $\mathsf{ProcOnion}$).

7: $\mathcal{B}$ sends O_1 to $\mathcal{A}$.

8: $\mathcal{B}$ gives oracle access to $\mathcal{A}$ (again using $\mathcal{O}.\mathsf{Dec}$ to decrypt ciphertexts for Q_{S}).

Finally, $\mathcal{B}$ guesses the bit b' that $\mathcal{A}$ outputs.

The reduction works since $\mathcal{B}$ wins if $\mathcal{A}$ wins; otherwise, $\mathcal{A}$ would be able to break the collision-resistance of the hash function. Clearly, the reduction runs in polynomial-time.

Hybrid^3—**swap k_j for fake key "$0 \ldots 0$"** Let Hybrid^3 be the same procedure as Hybrid^2 except in step 5, the challenger obtains ciphertext E_j by encrypting the all-zero key "$0 \ldots 0$" instead of key k_j:

5: form onion O_{j+1} (like in Hybrid^2);

form onion O_j by wrapping a layer around O_{j+1} (using key k_j produced from forming O_{j+1} but don't encrypt the key under $\mathsf{sk}(P_j)$):

$$\begin{aligned} & B_j^1 = \{P_{j+1}, E_{j+1}\}_{k_j} \\ \forall i \in \{2, \ldots, N-1\},\ & B_j^i = \{B_{j+1}^{i-1}\}_{k_j} \\ & E_j \leftarrow \mathsf{Enc}(\mathsf{pk}(P_j), h(B_j^1, \ldots, B_j^{N-1}), (0 \ldots 0)) \\ & C_j = \{C_{j+1}\}_{k_j}; \end{aligned}$$

form onion O_1 by wrapping layers around O_j (using keys produced from forming O_{j+1})

Here, we prove that $\mathcal{A}$ cannot distinguish between running Hybrid^2 and running Hybrid^3. For the sake of reaching a contradiction, suppose that $\mathcal{A}$ can distinguish between running Hybrid^2 (i.e., $b = 0$) and running Hybrid^3 (i.e., $b = 1$), then we can construct a reduction $\mathcal{B}$ that can break the CCA2-security of the underlying encryption scheme as follows:

1: $\mathcal{B}$ receives the router names $Q_{\mathsf{I}}, Q_{\mathsf{S}}$ from $\mathcal{A}$.

2: $\mathcal{B}$ generates keys $(\mathsf{pk}(Q_{\mathsf{S}}), \mathsf{sk}(Q_{\mathsf{S}}))$ for Q_{S} using the key generation algorithm G but gets the public key $\mathsf{pk}(Q_{\mathsf{I}})$ of Q_{I} from its challenger. $\mathcal{B}$ sends the public keys $(\mathsf{pk}(Q_{\mathsf{I}}), \mathsf{pk}(Q_{\mathsf{S}}))$ to $\mathcal{A}$.

3: $\mathcal{B}$ gives oracle access to $\mathcal{A}$; whenever $\mathcal{B}$ needs to process an onion $O = ((E, B^1, \ldots, B^{N-1}), C)$ for Q_{I}, $\mathcal{B}$ uses the decryption oracle $\mathcal{O}.\mathsf{Dec}$ to decrypt the ciphertext portion E of O. For all other "process onion" requests, $\mathcal{B}$ simply runs $\mathsf{ProcOnion}$.

4: $\mathcal{B}$ gets from $\mathcal{A}$ the challenge onion parameters: label ℓ, message m, forward path $P^{\rightarrow}$ and return path $P^{\leftarrow}$ and the public keys of the adversarial parties in the routing path. $\mathcal{B}$ sends the challenge messages $m^0 = k_j$ and $m^1 = (0 \ldots 0)$ to the challenger, and the challenger responds with the encryption E_j^b of one of the messages.

5: $\mathcal{B}$ forms onion O_{j+1} (like in Hybrid^1) and sets onion O_j to be $((E_j^b, B_j^1, \ldots, B_j^{N-1}), C_j)$

where

$$\begin{aligned} B_j^1 &= \{P_{j+1}, E_{j+1}\}_{k_j} \\ \forall i \in \{2, \ldots, N-1\},\ B_j^i &= \{B_{j+1}^{i-1}\}_{k_j} \\ C_j &= \{C_{j+1}\}_{k_j}. \end{aligned}$$

Finally, $\mathcal{B}$ forms onion O_1 by wrapping onion layers around O_j.

6: $\mathcal{B}$ modifies oracle $\mathcal{O}.\mathsf{PO_S}$ so that if running $\mathsf{ProcOnion}$ on an onion with header H_s outputs $(\mathsf{S}, (x, m'))$ for some label x and message m, $\mathcal{O}.\mathsf{PO_S}$ outputs $(\mathsf{S}, (\ell, m'))$ instead (all other "process onion" requests are handled by running $\mathsf{ProcOnion}$).

7: $\mathcal{B}$ sends O_1 to $\mathcal{A}$.

8: $\mathcal{B}$ gives oracle access to $\mathcal{A}$ (again using $\mathcal{O}.\mathsf{Dec}$ to decrypt ciphertexts for Q_{I}).

Finally, $\mathcal{B}$ guesses the bit b' that $\mathcal{A}$ outputs.

The reduction works since $\mathcal{B}$ wins if $\mathcal{A}$ wins; otherwise, $\mathcal{A}$ would be able to break the collision-resistance of the hash function. Clearly, the reduction runs in polynomial-time.

Hybrid^4—swap $(B_{j+1}^{N-j-1}, \ldots, B_{j+1}^{N-1})$ for truly random blocks Let Hybrid^4 be the same procedure as Hybrid^3 except in step 5, blocks $(N-j-1)$ to $(N-1)$ in O_{j+1} are formed using a truly random permutation function F rather than the PRP keyed with k_j:

5: form $\hat{O}_j = (\hat{H}_j, \hat{C}_j)$:

$$\begin{aligned} x &\leftarrow \mathcal{L}(1^\lambda) \\ (\hat{H}_j, \ldots, \hat{H}_d, k_j, \ldots, k_d, K_d) &\leftarrow \mathsf{FormHeader}(\rightarrow, x, (P_j, \ldots, P_d)) \\ ((\hat{C}_j, \ldots, \hat{C}_d), k_j, \ldots, k_d) &\leftarrow \mathsf{FormContent}(x, m, P^{\leftarrow}, k_j, \ldots, k_d, K_d); \end{aligned}$$

form $O_{j+1} = ((E_{j+1}, B_{j+1}^1, \ldots, B_{j+1}^{N-j-2}, R_{j+1}^{N-j-1}, \ldots, R_{j+1}^{N-1}), C_{j+1})$ from $\hat{O}_j = ((\hat{E}_j, \hat{B}_j^1, \ldots, \hat{B}_j^{N-1}), \hat{C}_j)$:

$$\begin{aligned} (P_{j+1}, E_{j+1}) &= \}\hat{B}_j^1\{_{k_j} \\ \forall i \in \{1, \ldots, N-j-2\},\ B_{j+1}^i &= \}\hat{B}_j^{i+1}\{_{k_j} \\ \forall i \in \{N-j-1, \ldots, N-2\},\ R_{j+1}^i &= F(\hat{B}_j^{i+1}) \\ C_{j+1} &= \}\hat{C}_j\{_{k_j}; \end{aligned}$$

form O_1 by wrapping layers around O_{j+1}

Here, we prove that $\mathcal{A}$ cannot distinguish between running Hybrid^3 and running Hybrid^4. For the sake of reaching a contradiction, suppose that $\mathcal{A}$ can distinguish between running Hybrid^3 (i.e., $b = 0$) and running Hybrid^4 (i.e., $b = 1$), then we can construct a reduction $\mathcal{B}$ that can break the underlying PRP-CCA security of the PRP as follows:

1: $\mathcal{B}$ receives the router names $Q_{\mathsf{I}}, Q_{\mathsf{S}}$ from $\mathcal{A}$.

2: $\mathcal{B}$ generates keys $(\mathsf{pk}(Q_{\mathsf{I}}), \mathsf{sk}(Q_{\mathsf{I}}))$ for Q_{I} and keys $(\mathsf{pk}(Q_{\mathsf{S}}), \mathsf{sk}(Q_{\mathsf{S}}))$ for Q_{S} using the key generation algorithm G and sends the public keys $(\mathsf{pk}(Q_{\mathsf{I}}, \mathsf{pk}(Q_{\mathsf{S}}))$ to $\mathcal{A}$.

3: $\mathcal{B}$ gives oracle access to $\mathcal{A}$.

4: $\mathcal{B}$ gets from $\mathcal{A}$ the challenge onion parameters: label ℓ, message m, forward path $P^{\rightarrow}$ and return path $P^{\leftarrow}$ and the public keys of the adversarial parties in the routing path.

5: $\mathcal{B}$ forms onion $O_j = ((E_j, B_j^1, \ldots, B_j^{N-1}), C_j)$ by running $\mathsf{FormHeader}(\rightarrow, x, (P_j, \ldots, P_d))$ and $\mathsf{FormContent}(x, m, k_j, \ldots, k_d, K_d)$. $\mathcal{B}$ queries the challenger for the pseudo-random permutations of $(B_j^2, \ldots, B_j^{N-j-1}$, and the challenger responds with $(B_{j+1}^1, \ldots, B_{j+1}^{N-j-2})$. $\mathcal{B}$ sets O'_{j+1} to be $((E_{j+1}, B_{j+1}^1, \ldots, B_{j+1}^{N-j-2}, B_{j+1}^{b,N-j-1}, \ldots, B_{j+1}^{b,N-1}), C_{j+1})$, where $B_{j+1}^i = \}B_j^{i+1}\{_{k_j}$ for all $i \in [N-j-2]$, and $C_{j+1} = \}C_j\{_{k_j}$. $\mathcal{B}$ forms onion O_1 by wrapping onion layers around O_{j+1}.

6: $\mathcal{B}$ modifies oracle $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ so that if running $\mathsf{ProcOnion}$ on an onion with header H_s outputs $(\mathsf{S}, (x, m'))$ for some label x and message m, $\mathcal{O}.\mathsf{PO}_{\mathsf{S}}$ outputs $(\mathsf{S}, (\ell, m'))$ instead (all other "process onion" requests are handled by running $\mathsf{ProcOnion}$).

7: $\mathcal{B}$ sends O_1 to $\mathcal{A}$.

8: $\mathcal{B}$ gives oracle access to $\mathcal{A}$.

Finally, $\mathcal{B}$ guesses the bit b' that $\mathcal{A}$ outputs.

The reduction works since the distribution of the input to $\mathcal{A}$ in steps 5-6 is exactly what is expected "in the wild". Clearly, the reduction runs in polynomial-time.

Hybrid^5—**swap** $(B_j^1, \ldots, B_j^{N-j-1})$ **and content** C_j **for truly random strings** Let Hybrid^5 be the same procedure as Hybrid^4 except in step 5, the first $N - j$ blocks and the content of onion O_j are outputs of a truly random permutation function F rather than outputs of the PRP keyed with k_j:

5: form onion O_{j+1} (with some truly random blocks);

form onion $O_j = ((E_j, R_j^1, \ldots, R_j^{N-j-1}, B_j^{N-j}, \ldots, B_j^{N-1}), R_j)$ by wrapping a layer around O_{j+1} (using key k_j produced from forming O_{j+1}):

$$\begin{aligned} & R_j^1 = F(P_{j+1}, E_{j+1}) \\ \forall i \in \{2, \ldots, N-j-1\},\ & R_j^i = F(B_{j+1}^{i-1}) \\ \forall i \in \{N-j, \ldots, N-1\},\ & B_j^i = \{B_{j+1}^{i-1}\}_{k_j} \\ & E_j \leftarrow \mathsf{Enc}(\mathsf{pk}(P_j), h(B_j^1, \ldots, B_j^{N-1}), (0 \ldots 0)) \\ & R_j = F(C_{j+1}); \end{aligned}$$

form onion O_1 by wrapping layers around O_j (using keys produced from forming O_{j+1})

Here, we prove that $\mathcal{A}$ cannot distinguish between running Hybrid^4 and running Hybrid^5. For the sake of reaching a contradiction, suppose that $\mathcal{A}$ can distinguish between running Hybrid^4 (i.e., $b = 0$) and running Hybrid^5 (i.e., $b = 1$), then we can construct a reduction $\mathcal{B}$ that can break the underlying pseudo-randomness of the PRP as follows:

1: $\mathcal{B}$ receives the router names $Q_{\mathsf{I}}, Q_{\mathsf{S}}$ from $\mathcal{A}$.

2: $\mathcal{B}$ generates keys $(\mathsf{pk}(Q_{\mathsf{I}}), \mathsf{sk}(Q_{\mathsf{I}}))$ for Q_{I} and keys $(\mathsf{pk}(Q_{\mathsf{S}}), \mathsf{sk}(Q_{\mathsf{S}}))$ for Q_{S} using the key generation algorithm G and sends the public keys $(\mathsf{pk}(Q_{\mathsf{I}}, \mathsf{pk}(Q_{\mathsf{S}}))$ to $\mathcal{A}$.

3: $\mathcal{B}$ gives oracle access to $\mathcal{A}$.

4: $\mathcal{B}$ gets from $\mathcal{A}$ the challenge onion parameters: label ℓ, message m, forward path $P^{\rightarrow}$ and return path $P^{\leftarrow}$ and the public keys of the adversarial parties in the routing path.

5: $\mathcal{B}$ forms onion O_{j+1} like in Hybrid^4 (with some truly random blocks). $\mathcal{B}$ sends to the challenger the sequence $((P_{j+1}, E_{j+1}), B_{j+1}^1, \ldots, B_{j+1}^{N-j-2}, C_{j+1})$. The challenger responds with $(E_j^b, B_j^{b,1}, \ldots, B_j^{b,N-j-1}, C_j^b)$ which are either pseudo-random permutations (if $b = 0$) or truly random permutations (if $b = 1$). $\mathcal{B}$ sets O_j to be $((E_j, B_j^{b,1}, \ldots, B_j^{b,N-j-1}, B_j^{N-j}, \ldots, B_j^{N-1}), C_j^b)$ where

$$\begin{aligned} \forall i \in \{N-j, \ldots, N-1\},\ & B_j^i = \{B_{j+1}^{i-1}\}_{k_j} \\ & E_j \leftarrow \mathsf{Enc}(\mathsf{pk}(P_j), t_j, (0 \ldots 0)), \end{aligned}$$

and where $t_j = h(B_j^{b,1}, \ldots, B_j^{b,N-j-1}, B_j^{N-j}, \ldots, B_j^{N-1})$. Finally, $\mathcal{B}$ forms onion O_1 by wrapping onion layers around O_j.

6: $\mathcal{B}$ modifies oracle $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ so that if running $\mathsf{ProcOnion}$ on an onion with header H_s outputs $(\mathsf{S}, (x, m'))$ for some label x and message m, $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ outputs $(\mathsf{S}, (\ell, m'))$ instead (all other "process onion" requests are handled by running $\mathsf{ProcOnion}$).

7: $\mathcal{B}$ sends O_1 to $\mathcal{A}$.

8: $\mathcal{B}$ gives oracle access to $\mathcal{A}$.

Finally, $\mathcal{B}$ guesses the bit b' that $\mathcal{A}$ outputs.

The reduction works since the distribution of the input to $\mathcal{A}$ in steps 5-6 is exactly what is expected "in the wild". Clearly, the reduction runs in polynomial-time.

Hybrid^6—**swap onion for intermediary P_j for onion for recipient P_j** Let Hybrid^6 be the same procedure as Hybrid^5 except in steps 5-6, the challenger wraps layers around a bogus onion O_j for recipient P_j to obtain the output O_1 and modifies oracle $\mathcal{O}.\mathsf{PO}_\mathsf{I}$ accordingly:

5: form onion O_{j+1} (with some truly random blocks);

form bogus onion $O_j = ((E_j, R_j^1, \ldots, R_j^{N-j-1}, B_j^{N-j}, \ldots, B_j^{N-1}), R_j)$ for recipient P_j:

$$\begin{aligned} \forall i \in [N-j-1],\ & R_j^i \leftarrow_\$ \{0,1\}^{L_1(\lambda)} \\ & k_1, \ldots, k_N \leftarrow_\$ \{0,1\}^{|k|(\lambda)} \\ \forall i \in \{N-j, \ldots, N-1\},\ & B_j^i = \}\ldots\}0\ldots0\{_{k_{N-i-1}}\{_{k_{N-j}} \\ & E_j \leftarrow \mathsf{Enc}(\mathsf{pk}(P_j), h(B_j^1, \ldots, B_j^{N-1}), (0\ldots0)) \\ & R_j \leftarrow_\$ \{0,1\}^{L_1(\lambda)}; \end{aligned}$$

form onion O_1 by wrapping layers around O_j' (using keys $k_1, \ldots, k_{N-1}$)

6: modify $\mathcal{O}.\mathsf{PO}_\mathsf{I}$: to "process" an onion $O = (H_j, C)$ with the same header H_j as O_j, $\mathcal{O}.\mathsf{PO}_\mathsf{I}$ returns $(\mathsf{I}, (\mathsf{CompleteOnion}(1^\lambda, \mathsf{pp}, H_{j+1}, C), P_{j+1})$;
modify $\mathcal{O}.\mathsf{PO}_\mathsf{S}$: to "process" an onion O with the header H_s (i.e., last header in $H^{\leftarrow}$), $\mathcal{O}.\mathsf{PO}_\mathsf{S}$ extracts the message $m' = \mathsf{RecoverReply}(\lambda, \mathsf{pp}, O, P_j, \mathsf{sk}(P_j))$ and returns $(\mathsf{S}, (\ell, m'))$

The adversary can query the oracle $\mathcal{O}.\mathsf{PO}_\mathsf{I}$ to process an onion with the correct challenge header but with "mangled" content. In this case, the peeled onion in Hybrid^5 looks like the peeled onion in Hybrid^6 because the former has a truly random header and content whereas the latter has a truly random header and pseudo-random content. For all other queries, the

responses in the hybrids are statistically the same. Thus, using a straightforward hybrid argument, we can show that the adversary cannot distinguish between running Hybrid^5 and running Hybrid^6.

Hybrid^7—**swap truly random blocks and content in O_j for pseudo-random blocks $(B_j^1, \ldots, B_j^{N-j-1}, C_j)$** Let Hybrid^7 be the same procedure as Hybrid^6 except in step 5, the challenger wraps layers around the onion O_j with real blocks and real content to obtain the output O_1:

5: form onion O_{j+1} (with some truly random blocks);

form onion O_j for recipient P_j:

$$\begin{aligned} y &\leftarrow_{\$} \mathcal{M}(\lambda) \\ ((((\hat{E}_j, B_j^1, \ldots, B_j^{N-1}), C_j)), (), \kappa) &\leftarrow \mathsf{FormOnion}((0 \ldots 0), y, (P_j), \mathsf{pk}(P_j), (), ()) \\ E_j &\leftarrow \mathsf{Enc}(\mathsf{pk}(P_j), h(B_j^1, \ldots, B_j^{N-1}), (0 \ldots 0)); \end{aligned}$$

form onion O_1 by wrapping layers around O_j (using keys from $\mathsf{FormOnion}$)

$\mathcal{A}$ cannot distinguish between running Hybrid^6 and running Hybrid^7. Otherwise, we could construct a reduction (very similar to the reduction used for proving that $\mathsf{Hybrid}^4 \approx \mathsf{Hybrid}^5$) that can break the underlying pseudo-randomness of the PRP.

Hybrid^8—**swap truly random blocks in H_{j+1} for pseudo-random blocks $(B_{j+1}^{N-j-1}, \ldots, B_{j+1}^{N-1})$** Let Hybrid^8 be the same procedure as Hybrid^7 except in step 5, the challenger wraps layers around a real onion O_j to obtain the output O_1:

5: form onion O_{j+1} (with all pseudo-random blocks);

form onion O_j for recipient P_j;

$$\begin{aligned} y &\leftarrow_{\$} \mathcal{M}(\lambda) \\ ((((\hat{E}_j, B_j^1, \ldots, B_j^{N-1}), C_j)), (), \kappa) &\leftarrow \mathsf{FormOnion}((0 \ldots 0), y, (P_j), \mathsf{pk}(P_j), (), ()) \\ E_j &\leftarrow \mathsf{Enc}(\mathsf{pk}(P_j), h(B_j^1, \ldots, B_j^{N-1}), (0 \ldots 0)); \end{aligned}$$

form onion O_1 by wrapping layers around O_j

$\mathcal{A}$ cannot distinguish between running Hybrid^7 and running Hybrid^8. Otherwise, we could construct a reduction (very similar to the reduction used for proving that $\mathsf{Hybrid}^3 \approx \mathsf{Hybrid}^4$) that can break the underlying PRP-CCA2 security of the PRP.

Hybrid^9—**swap key "$0\ldots0$" for for real key** k_j Let Hybrid^9 be the same procedure as Hybrid^8 except in step 5, the challenger wraps layers around a real onion O_j to obtain the output O_1:

5: form onion O_{j+1};

form onion O_j for recipient P_j;

form onion O_1 by wrapping layers around O_j

$\mathsf{Hybrid}^8 \approx \mathsf{Hybrid}^9$ $\mathcal{A}$ cannot distinguish between running Hybrid^8 and running Hybrid^9. Otherwise, we could construct a reduction (very similar to the reduction used for proving that $\mathsf{Hybrid}^2 \approx \mathsf{Hybrid}^3$) that can break the underlying CCA2-security of the encryption scheme.

Finally, Hybrid^9 and $\mathsf{Experiment}^1$ produce the same result. In both procedures, onion O_1 is formed using the all-zero label "$0\ldots0$", a random message y, the truncated path $(P_1, \ldots, P_j)$ as the forward path and the empty return path "()", and the oracle $\mathcal{O}.\mathsf{PO}_\mathsf{I}$ ensures that O_j "peels" to the separately formed O_{j+1}.

This concludes our proof for case (a). The proofs for cases (b) and (c) are similar. □

We now show that the history of outputs that $\mathcal{S}$ produces is indistinguishable from one produced by running the real protocol to any environment $\mathcal{Z}$.

Theorem 11. *Shallot Encryption Scheme (Section 4.4) SUC-realizes the ideal functionality $\mathcal{F}_{\mathsf{ROES}}$ (Definition 15).*

Proof. From Lemma 12, it suffices to prove that if a repliable-onion encryption scheme Σ is repliable-onion secure, then it also SUC-realizes the ideal functionality $\mathcal{F}_{\mathsf{ROES}}$.

Our proof is via a hybrid argument.

Let $\mathsf{Experiment}^1$ be the ideal onion routing protocol in which the ideal honest parties query $\mathcal{F}_{\mathsf{ROES}}$ to form onions, process onions and form return onions.

Let $\mathsf{Experiment}^0$ be the real onion routing protocol in which the honest participants run Σ's algorithms to form onions, process onions and form return onions.

Let Hybrid^0 be the experiment with the same set up at $\mathsf{Experiment}^0$ except that the challenger controls all honest parties and the ideal functionality $\mathcal{F}_{\mathsf{ROES}}$, and the environment is the adversary.

We define the remaining hybrid experiments as follows:

Let numFO be the upper bound on the number of honest "form onion" queries from the environment.

For all $i \in [\mathsf{numFO}]$, let Hybrid^i be the experiment in which the first i "form onion" queries are "simulated", and all remaining queries are "real". The challenger runs $\mathcal{F}_{\mathsf{ROES}}$ for the first i "form onion" queries and runs $\mathsf{FormOnion}$ for the remaining ones. The challenger always runs $\mathcal{F}_{\mathsf{ROES}}$ to process an onion or to form a reply. This works because $\mathcal{F}_{\mathsf{ROES}}$ just runs $\mathsf{ProcOnion}$ or $\mathsf{FormReply}$ in cases where $\mathcal{F}_{\mathsf{ROES}}$ doesn't recognize the query onion's header.

Recall that N is the upper bound on the length of the forward (or return) path which is also an upper bound on the number of segments per "form onion" query.

Next, we describe the hybrids between Hybrid^{i-1} and Hybrid^i. For all $j \in [N]$, let $\mathsf{Hybrid}^{i,j}$ be the experiment in which the first $i-1$ "form onion" queries and the first j segments of the i-th "form onion" query are "simulated", and all remaining segments are "real". (i) The first $i-1$ "form onion" queries are processed via $\mathcal{F}_{\mathsf{ROES}}$. (ii) All queries after the i-th query are processed by running $\mathsf{ProcOnion}$. (iii) The i-th query is processed specially as follows.

Let $(P^{\rightarrow}, P^{\leftarrow})$ denote the routing path of the i-th query. First, the challenger partitions $(P^{\rightarrow}, P^{\leftarrow})$ into (at most) $j+1$ subpaths $(p_1^{\rightarrow}, \ldots, p_j^{\rightarrow}, q^{\rightarrow})$, consisting of the first j segments $(p_1^{\rightarrow}, \ldots, p_j^{\rightarrow})$ of $(P^{\rightarrow}, P^{\leftarrow})$ (or as many as they are) and the remaining subpath $q^{\rightarrow}$ of $(P^{\rightarrow}, P^{\leftarrow})$ not covered by the segments (if it exists).

To process the i-th "form onion" request, the challenger essentially runs the same code as $\mathcal{F}_{\mathsf{ROES}}$ except with the subpaths as the "segments" of $(P^{\rightarrow}, P^{\leftarrow})$. (Onion layers and return paths are stored in the same dictionaries, $\mathsf{OnionDict}$ and $\mathsf{PathDict}$, used by the unmodified $\mathcal{F}_{\mathsf{ROES}}$ code.) As before, the challenger always runs $\mathcal{F}_{\mathsf{ROES}}$ to process an onion or to form a reply.

By construction, (i) $\mathsf{Experiment}^0$ and Hybrid^0 produce identical results, (ii) for all $i \in [\mathsf{numFO}]$, $\mathsf{Hybrid}^{i-1,N}$ and Hybrid^i produce identical results, and (iii) $\mathsf{Hybrid}^{\mathsf{numFO},N}$ and $\mathsf{Experiment}^1$ produce identical results. For any $i \in [\mathsf{numFO}]$ and $j \in [N-1]$, the repliable-onion security of Σ guarantees that the environment cannot distinguish between running $\mathsf{Hybrid}^{i,j}$ and $\mathsf{Hybrid}^{i,j+1}$. Since the total number of segments is polynomially bounded in the security parameter, it follows that the environment cannot distinguish between running $\mathsf{Experiment}^0$ and running $\mathsf{Experiment}^1$. In other words, Σ SUC-realizes $\mathcal{F}_{\mathsf{ROES}}$. □

4.6 Concluding remarks

In this chapter, we gave the first ideal functionality, $\mathcal{F}_{\mathsf{ROES}}$, for repliable onion encryption and the first onion encryption scheme, Shallot Encryption Scheme, proven to UC-realize

$\mathcal{F}_{\mathsf{ROES}}$.

Bibliography

[AAC+11] Mário S. Alvim, Miguel E. Andrés, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. Differential privacy: on the trade-off between utility and information leakage. In *FAST 2011*, pages 39–54. Springer, 2011.

[Abe98] Masayuki Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 437–447. Springer, Heidelberg, May / June 1998.

[AL20] Megumi Ando and Anna Lysyanskaya. Cryptographic shallots: A formal treatment of repliable onion encryption. *IACR Cryptology ePrint Archive*, 2020: 215, 2020. URL `https://eprint.iacr.org/2020/215`.

[ALU18] Megumi Ando, Anna Lysyanskaya, and Eli Upfal. Practical and provably secure onion routing. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 144:1–144:14. Schloss Dagstuhl, July 2018.

[ALU19] Megumi Ando, Anna Lysyanskaya, and Eli Upfal. On the complexity of anonymous communication through public networks. *CoRR*, abs/1902.06306, 2019. URL `http://arxiv.org/abs/1902.06306`.

[Ber04] Berman, Ron and Fiat, Amos and Ta-Shma, Amnon. Provable Unlinkability against Traffic Analysis. In *International Conference on Financial Cryptography*, pages 266–280. Springer, 2004.

[BFTS04] Ron Berman, Amos Fiat, and Amnon Ta-Shma. Provable unlinkability against traffic analysis. In Ari Juels, editor, *FC 2004*, volume 3110 of *LNCS*, pages 266–280. Springer, Heidelberg, February 2004.

[BGKM12] Michael Backes, Ian Goldberg, Aniket Kate, and Esfandiar Mohammadi. Provably secure and practical onion routing. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 369–385. IEEE, 2012.

[BKM+13] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. Anoa: A framework for analyzing anonymous communication protocols. In *Computer Security Foundations Symposium (CSF), 2013 IEEE 26th*, pages 163–178. IEEE, 2013.

[BS15] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.2*, 2015.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CB95] David A. Cooper and Kenneth P. Birman. Preserving privacy in a network of mobile computers. In *1995 IEEE Symposium on Security and Privacy*, pages 26–38. IEEE Computer Society Press, 1995.

[CBM15] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015.

[Cha81] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cha88a] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, January 1988.

[Cha88b] Chaum, David. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[Chr20] Miranda Christ. New lower bounds on the complexity of provably anonymous onion routing. Undergraduate honors thesis, Brown University, Providence, RI 02912 USA, 2020.

[CL05] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 169–187. Springer, Heidelberg, August 2005.

[Coo95] Cooper, David A and Birman, Kenneth P. Preserving Privacy in a Network of Mobile Computers. In *Proceedings of the Sixteenth IEEE Symposium on Security and Privacy*, pages 26–38. IEEE, 1995.

[Cor15] Corrigan-Gibbs, Henry and Boneh, Dan and Mazières, David. Riposte: An Anonymous Messaging System Handling Millions of Users. In *Proceedings of the Thirty-sixth IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[Cot95] Lance Cottrell. Mixmaster and remailer attacks, 1995.

[CPP08] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Information and Computation*, 206(2–4):378–401, 2008.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Heidelberg, August 1998.

[CSV16] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 265–296. Springer, Heidelberg, March 2016.

[Czu99] Czumaj, A and Kanarek, P and Kutyłowski, M and Lorys, K. Distributed Stochastic Processes for Generating Random Permutations. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 99, pages 271–280, 1999.

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *2003 IEEE Symposium on Security and Privacy*, pages 2–15. IEEE Computer Society Press, May 2003.

[DG09] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *2009 IEEE Symposium on Security and Privacy*, pages 269–282. IEEE Computer Society Press, May 2009.

[DL04] George Danezis and Ben Laurie. Minx: A simple and efficient anonymous packet format. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 59–65, 2004.

[DMMK18] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two. In *2018 IEEE Symposium on Security and Privacy*, pages 108–126. IEEE Computer Society Press, May 2018.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320, 2004.

[Dod04] Dodis, Yevgeniy and Reyzin, Leonid and Smith, Adam. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 523–540. Springer, 2004.

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4): 211–407, 2014.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Heidelberg, May 2004.

[Edg17] Timothy H Edgar. *Beyond Snowden: Privacy, Mass Surveillance, and the Struggle to Reform the NSA*. Brookings Institution Press, 2017.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 218–229. ACM, 1987.

[Gol98] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001. ISBN 0-521-79172-3 (hardback). xix + 372 pp. LCCN QA268.G5745 2001.

[GT96] Ceki Gulcu and Gene Tsudik. Mixing e-mail with babel. In *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*, pages 2–16. IEEE, 1996.

[HS05] Don Hush and Clint Scovel. Concentration of the hypergeometric distribution. *Statistics & Probability Letters*, 75(2):127–132, 2005.

[Jen13] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media, 2013.

[JWJ+13] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul F. Syverson. Users get routed: traffic correlation on tor by realistic adversaries. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 337–348. ACM Press, November 2013.

[KBS19] Christiane Kuhn, Martin Beck, and Thorsten Strufe. Breaking and (partially) fixing provably secure onion routing. *arXiv preprint arXiv:1910.13772*, 2019.

[KCDF17] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 406–422, 2017.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014.

[MC00] Ulf Möller and Lance Cottrell. Mixmaster protocol?version 2. unfinished draft, Jan 2000.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[ØS06] Lasse Øverlier and Paul Syverson. Locating hidden servers. In *2006 IEEE Symposium on Security and Privacy*, pages 100–114. IEEE Computer Society Press, May 2006.

[Par96] Sameer Parekh. Prospects for remailers. *First Monday*, 1(2), 1996.

[Rac93] Rackoff, Charles and Simon, Daniel R. Cryptographic Defense Against Traffic Analysis. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, pages 672–681. ACM, 1993.

[Ray01] Raymond, Jean-François. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Designing Privacy Enhancing Technologies*, pages 10–29. Springer, 2001.

[RS93] Charles Rackoff and Daniel R. Simon. Cryptographic defense against traffic analysis. In *25th ACM STOC*, pages 672–681. ACM Press, May 1993.

[SEV+15] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: Routing Attacks on Privacy in Tor. In *USENIX Security Symposium*, pages 271–286, 2015.

[TGL+17] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: a distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 423–440, 2017.

[vdHLZZ15] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles, Monterey, CA, USA, October 4-7, 2015*, pages 137–152, 2015.

[WSJ+18] Ryan Wails, Yixin Sun, Aaron Johnson, Mung Chiang, and Prateek Mittal. Tempest: Temporal dynamics in anonymity systems. *PoPETs*, 2018(3):22–42, 2018. URL `https://doi.org/10.1515/popets-2018-0019`.