Poly Onions: Achieving Anonymity in the Presence of Churn

Megumi Ando* Miranda Christ† Anna Lysyanskaya‡ Tal Malkin†

March 25, 2022

Abstract

Onion routing is a popular approach towards anonymous communication. Practical implementations are widely used (for example, Tor has millions of users daily), but are vulnerable to various traffic correlation attacks, and the theoretical foundations, despite recent progress, still lag behind. In particular, all works that model onion routing protocols and prove their security only address a single run, where each party sends and receives a single message of fixed length, once. Moreover, they all assume a static network setting, where the parties are stable throughout the lifetime of the protocol. In contrast, real networks have a high rate of churn (nodes joining and exiting the network), real users want to send multiple messages, and realistic adversaries may observe multiple runs of the protocol.

In this paper, we initiate a formal treatment of onion routing in a setting with multiple runs over a dynamic network with churn. We provide the following contributions.

1. We define the cryptographic primitive of *poly onion encryption*, which is appropriate for a setting with churn. This primitive is inspired by duo onions, introduced by Iwanik, Klonowski, and Kutylowski (Communications and Multimedia Security, 2005) towards improving onion delivery rate. We generalize the idea, change it to add auxiliary helpers towards supporting better security, and propose formal definitions.
2. We construct an instantiation of poly onion encryption based on standard cryptographic primitives (CCA secure public key encryption with tags, PRP, MAC, and secret sharing). Our construction is secure against an active adversary, and is parameterized to allow flexible instantiations supporting a range of corruption thresholds and churn limits.
3. We formally model anonymous onion routing for multiple runs in the setting with churn, including a definition of strong anonymity, where the adversary has CCA-like access to oracles for generating and processing onions.
4. We prove that if an onion routing protocol satisfies a natural condition we define ("simulatability"), then strong single-run anonymity implies strong multiple-run anonymity. This condition is satisfied by existing onion routing schemes, such as the Π_p protocol of Ando, Lysyanskaya, and Upfal (ICALP 2018). As a consequence, these schemes are anonymous also for multiple runs (although not when there is churn).
5. We provide an anonymous routing protocol, "Poly Π_p," and prove that it is anonymous in the setting with churn, against a passive adversary. We obtain this construction by using an instance of our poly onion encryption within the Π_p protocol.

*MITRE, mando@mitre.org

†Columbia University, {mchrist,tal}@cs.columbia.edu

‡Brown University, anna@cs.brown.edu

Contents

1 Introduction **1**
- 1.1 Our contributions . . . 2
- 1.2 Related work . . . 5

2 Modeling the problem **5**

3 Onion encryption for churn **7**
- 3.1 I/O syntax . . . 7
- 3.2 Correctness . . . 8
- 3.3 Security . . . 9

4 Our poly onion encryption scheme **11**
- 4.1 Overview of Poly Onion Encryption . . . 12
  - 4.1.1 Anatomy of an onion . . . 12
  - 4.1.2 Overview of processing an onion. . . . 13
- 4.2 Forming an onion . . . 13
  - 4.2.1 Wrapping an onion . . . 14
- 4.3 Processing an onion . . . 15
- 4.4 Analysis of Poly Onion Encryption . . . 15
- 4.5 Generalizing to more than two candidate processing parties . . . 17
- 4.6 Correctness of Poly Onion Encryption with multiple candidates . . . 17

5 Anonymity in the setting with churn **18**
- 5.1 Definitions of anonymity . . . 18
- 5.2 Simulatable onion routing protocols . . . 20
- 5.3 From single-run to multi-run anonymity . . . 22

6 Multi-run strongly anonymous onion routing **23**
- 6.1 Poly Π_p is multi-run anonymous in the presence of churn . . . 24

A Security of Poly Onion Encryption **29**
- A.1 $\mathsf{Hybrid}^1 \approx \mathsf{Hybrid}^2$ . . . 29
- A.2 $\mathsf{Hybrid}^2 \approx \mathsf{Hybrid}^3$ . . . 29
- A.3 $\mathsf{Hybrid}^3 \approx \mathsf{Hybrid}^4$ . . . 32
- A.4 $\mathsf{Hybrid}^4 \approx \mathsf{Hybrid}^5$ . . . 32
- A.5 $\mathsf{Hybrid}^5 \approx \mathsf{Hybrid}^6$ . . . 34
- A.6 $\mathsf{Hybrid}^6 \approx \mathsf{Hybrid}^7$ . . . 34
- A.7 $\mathsf{Hybrid}^7 \approx \mathsf{Hybrid}^8$ . . . 34

1 Introduction

Anonymous communication. Privacy is a fundamental human right, and it is increasingly under threat. We need to be able to connect to our desired websites and communicate with each other privately without being subject to scrutiny and interference, and, in some cases — such as when dissidents are trying to help each other in an oppressive regime — physical threats. While encryption provides confidentiality of message content, much information can still be gleaned from observed traffic patterns in a network, revealing such information as who is communicating with whom, when, and for how long.

Our goal is to implement anonymous channels over a point-to-point network, such as the Internet. Specifically, we want every user to be able to send a message to another user so that an adversary monitoring the network and controlling (passively or actively) a fraction of its nodes, possibly including the recipient of the message, should not be able to tell who is communicating with whom. That is, the scenario in which Alice sends a message to Bob should be indistinguishable from the one in which she sends one to Carol, instead.

How can we achieve such anonymous communication? A trivial solution is to use a secure computation protocol among all parties, where each party inputs their message and destination, and the functionality delivers the messages (where the output of each party is the set of all messages sent to that party, in lexicographic order). This solution is clearly not adequate: it is extremely inefficient and involves massive communication among the parties, all of whom should be available and interact back and forth throughout the protocol run. A few other approaches towards achieving anonymity have been proposed, but the gap between what is needed and the existing solutions remains large. In this paper, we focus on bridging this gap, working within the onion routing framework.

Onion routing. Onion routing [10, 15, 19] is a popular approach towards achieving anonymity. The basic idea is that when Alice wants to send a message to Bob, she chooses random intermediate nodes constituting a path from her to Bob. She then prepares a cryptographic object called an "onion," which consists of layered ciphertexts, with one layer per node on the path. Alice then sends the onion through the path, with each intermediate party "peeling" a layer of the onion to discover the next node on the path until the onion reaches its destination. When several onions are peeled by the same honest intermediary in the same round, the adversary cannot correlate the incoming onions with the outgoing ones; we refer to this as "mixing." Thanks to mixing, it is possible to expect anonymity with onion routing. Tor ("The onion router") is the most widely used anonymity network, consisting of thousands of routers and used by millions of users daily [19]. While clearly practical, it is also vulnerable to traffic correlation attacks [26, 29, 31].

Starting with [9], in recent years there have been several works attempting to put onion routing on a solid theoretical foundation. For example, we know that sufficiently shuffling the onions provides anonymity from the passive adversary [3] and that a polylog number of rounds is both necessary (e.g., [12,17,18]) and sufficient (e.g., [3,27]) for this. Providing anonymity from the active adversary is significantly more challenging than shuffling. Surprisingly, a polylog number of rounds is still sufficient for achieving anonymity in the active adversary setting with fault tolerance [3]. Several other works address only onion construction without discussing or analyzing the onion routing protocol (e.g., [2, 22]).

However, the theoretical modeling, while solving important challenges, is still quite far from what we need in practice. Perhaps the most glaring issue is the fact that all the works that model and analyze onion routing protocols only address a *single* instance of message routing for a restricted set of communication patterns. Specifically, each party is instructed to send a (fixed length) message to another party such that everyone sends a message and everyone receives a message, and the protocol for communicating all messages only occurs once. This is in contrast to real-world scenarios where parties send messages of varying lengths and many times without coordinating with other parties. An additional challenge for a system that supports ongoing traffic is *network churn*: the nodes on the network may go offline or join back. Realistic networks have high rates of churn, but this has not been addressed by the above works. Other recent protocols addressing anonymity [24, 32, 33] also operate only in the static network setting (without any nodes joining or exiting the network).

Network churn. Onion routing schemes rely on communication through intermediate nodes, which should be known at the time the onion is created. However, practical networks are dynamic, allowing for significant node churn. For example, measurement studies of real-world P2P networks [20, 30] show that the churn rate is quite high: nearly 50% of peers in real-world networks can be replaced within an hour.

Churn poses significant challenges for anonymous routing, even at the definitional level. One obvious issue is that in standard onion routing, the entire route of an onion is chosen in advance when the onion is created. Thus, if only just one of the parties on the route churns out and goes offline, the onion is lost. We note that this is a problem not only with correctness and reliability, but also with security/anonymity. Indeed, if an adversary observes an onion originating with Alice and going to an offline node (hence dropped), and then sees that Bob ended up receiving fewer onions than other parties, the adversary may conclude that the dropped onion from Alice was likely intended for Bob.

1.1 Our contributions

In this work we initiate the formal treatment of onion routing in a setting with multiple runs over a dynamic network with churn.

Poly onions. A natural idea towards overcoming churn is to construct an onion in a way that allows for more than one option for each hop on the route. This way the onion will not be dropped if one intermediate node is offline, and can instead be routed to a backup intermediary. This idea was put forward by Iwanik, Klonowski, and Kutyłowski [21], who suggested "duo onion" encryption as a way to improve onion delivery rates when there is network churn. A duo onion has two candidate intermediary servers for each onion layer. If the first candidate is offline, the onion can be sent to the second candidate. While Iwanik et al. proposed a duo onion construction and did a back-of-the-envelope analysis of its efficiency, they did not formalize duo onion routing nor prove the security of this scheme. In fact, duo onions may be less secure than regular onion routing because the adversary has the ability to influence the path of the onion in a way that may allow it to trace it through the network.

Our first contribution (Definitions 1 and 2) is a formal definition of *poly onion encryption*. Poly onion encryption is inspired by duo onion encryption described above, but it does not suffer from the same security flaw.

Our definition introduces auxiliary parties, called *helpers*, for each hop in the routing path. The helpers can ensure that an onion is routed to its backup intermediary for the next hop only if the preferred one is offline. This fixes the flaw in duo onions: a corrupted party can no longer choose any candidate it wishes in the next round. It also makes onion encryption more complicated: processing a poly onion is an interactive protocol involving the current intermediate node, the candidates for where to send it next, and the helpers.

In part because of this interactivity, defining correctness and security for a poly onion requires some care. Intuitively, correctness (Definition 1) captures the requirement that the onion will reach its intended destination, and the intended message will be recovered, as long as some condition holds (for standard onion routing, the condition is that every intermediate party on the path behaves honestly). Security (Definition 2) captures the requirement that the adversary will not be able to correlate an input onion with one of several output onions coming out of a processing party, as long as some condition holds (for standard onion routing, the condition is that the processing party is honest). These conditions can be complex, since they depend on who is online, and each hop involves many potential parties (candidates and helpers). We capture these conditions through correctness and security predicates. (Note that different poly onion encryption schemes may be correct/secure with respect to different predicates.)

Our second contribution is a construction of poly onions from standard cryptographic primitives: CCA secure public-key encryption with tags [13,14], PRP, MAC, and secret sharing. Our construction, Poly Onion Encryption (Section 4), is parameterized in terms of the number of candidates κ, the size ν of the helper committee, and the secret sharing reconstruction threshold α.

In our construction, the committee members are responsible for ensuring that a processing party indeed sends its onion to the first online candidate in its list. At a high level, an onion is valid only if it comes with a key header used for processing it. An onion typically comes with a key header encrypted for the first candidate in its next hop. If the first candidate is offline, the processing party must enlist the help of the committee to construct this key header for an alternate candidate. For this purpose, each onion comes with inputs for the onion processing protocol. The processing party distributes these inputs to the committee members, who check that the first candidate is indeed offline and select the next online candidate in the list. The committee members return secret shares; given at least $\alpha \cdot \nu$ shares the processing party reconstructs the header for the alternate candidate.

We prove that our construction is correct and secure against an active adversary, with respect to corresponding predicates that we define. The correctness predicate for each step roughly requires that the processing party is honest and online, and that at most $\alpha \cdot \nu$ members of the committee are corrupted. The security predicate roughly requires that there are no corrupted parties appearing before the first honest and online party in the list of next candidates, and that fewer than $\alpha \cdot \nu$ members of the committee are corrupted. This is analogous to the condition for standard onions (where the processing party is required to be honest), so our predicate is only minimally stronger than that of standard onion encryption. As long as enough parties overall are honest, and committees are chosen randomly, we can increase the committee size to boost the probability that fewer than $\alpha \cdot \nu$ committee members are corrupted. These parameters can be instantiated so that the predicates are satisfied with high enough probability to achieve anonymity in the overall onion routing protocol, discussed below.

Anonymous onion routing. Let us revisit why achieving anonymity in the presence of churn is difficult. As an illustrative example, consider the simple onion routing protocol Π_p [3]. In Π_p, each sender routes an onion randomly through a network of server nodes such that the onion mixes with a polylog number of onions, a polylog number of times. It was shown that Π_p is anonymous from the passive adversary who corrupts up to a constant fraction of the servers (in the static setting) just by shuffling the onions in this way. However, it is not anonymous when we add churn to the equation: if the adversary observes that Alice's onion churns out before it gets a chance to mix with too many onions, then she may be able to infer who Alice's recipient is by observing who doesn't receive an onion at the end of the protocol. Intuitively, Iwanik et al.'s duo onion idea [21] is a partial solution to this problem; it is more difficult for Alice's onion to churn out if, at each hop, it can route to an alternative random server if the preferred one is offline. However, duo onions (without helper parties) don't necessarily mix at honest servers. This is, in part, because an adversarial intermediary P_i may behave honestly and route Alice's onion to the honest preferred next server P^+_{i+1} but still learn what the peeled onion looks like if the alternative next server P^-_{i+1} is adversarial: in this case, P_i knows what the alternative onion O^-_{i+1} for P^-_{i+1} looks like, and P^-_{i+1} knows how to peel it. The purpose of having helper parties in poly onions is to prevent this. Thus, a natural question is: can we make Π_p anonymous in the setting with churn by using poly onions?

Before we could answer this question, it was necessary to first define what it means for an onion routing protocol to be anonymous in the presence of churn. Prior definitions of anonymity are defined only for a single protocol run in the static setting, whereas most applications operate over multiple runs over a long period of time, and so we should model them as operating (concurrently with other runs) in a dynamic setting. For our third contribution, we present a definition of anonymity for the multi-run setting with churn.

Our definition of anonymity (Definition 3) is as follows. An onion routing protocol is *anonymous* if the adversary cannot tell whether it is interacting with the challenger over L runs on input vectors $\sigma^0_1, \ldots, \sigma^0_L$ or on input vectors $\sigma^1_1, \ldots, \sigma^1_L$. It is *strongly* anonymous if the adversary can query the challenger to peel onions before and after the L challenge runs. It is *adaptively* anonymous if the adversary chooses the inputs and who is online/offline for the i^{th} run based on the prior $i-1$ runs. The strongest definition that is most helpful in an operational setting is the strong multi-run adaptive version of the definition.

Our fourth contribution is to show (Theorem 2) that for a large class of onion routing protocols, which we call simulatable, multi-run anonymity in the static (resp. dynamic) setting is equivalent to single-run anonymity in the static (resp. dynamic) setting. Informally, a *simulatable* onion routing protocol (Definition 4) is one where the adversary cannot tell whether it is interacting in the real setting in which the challenger runs the protocol on behalf of honest parties using the honest parties' secret keys, or in the ideal setting in which the challenger fakes the run without knowledge of the honest parties' secret keys. Since most onion routing protocols are simulatable, including Π_p, an immediate consequence is that practical onion routing protocols that satisfy multi-run anonymity exist – albeit in the static setting without churn.

Armed with a definition of anonymity for the dynamic setting and Theorem 2, we answer our question about Π_p in the affirmative. For our final contribution, we present a new onion routing protocol, Poly Π_p, that uses Poly Onion Encryption instead of standard onion encryption and prove that it satisfies (strong) multi-run (adaptive) anonymity when fewer than half of the parties can be offline or (passively) corrupted.

Open problems. Our work makes significant progress towards bridging the gap between theoretical foundations of onion routing, and required anonymity in realistic settings. Still, some important problems remain open.

First, can we achieve anonymous routing with churn against active adversaries? Note that Poly Π_p already provides protection against some types of active malicious behavior: poly onion encryption is secure against active adversaries, and Poly Π_p is also secure against an adversary that decides the churn schedule. However, anonymity breaks when an active adversary can selectively drop onions in a more "adaptive" way, namely in the middle of a run. We also note that in the static setting, the protocol $\Pi_{\bowtie}$ of Ando, Lysyanskaya and Upfal [4] achieves anonymity even against active adversaries. However, simply replacing their onions with our poly onions would still not yield a scheme that is anonymous in the setting with churn, because used as is, their protocol would equate churn with malicious activity and simply not work; an added complication is that $\Pi_{\bowtie}$ is not simulatable.

A second open problem is to address more general communication patterns. As in prior work [3, 4, 32, 33], a single run of our protocol is also restricted to the so called "Simple I/O" setting, where each party sends and receives exactly one message of a fixed length. The fact that we address the multi-run case partially mitigates the issue, as it provides a way to handle longer messages, by breaking them to several runs. Nonetheless, the assumption that the communication pattern in each run is a permutation is still limiting. Defining what anonymity should mean for more general communication patterns, which patterns can be efficiently supported anonymously, and how to construct such protocols, is a challenging and interesting topic for future work.

1.2 Related work

To the best of our knowledge, all existing onion routing works either (i) do not have any theoretical modeling or provable security, or deviate from standard notions of indistinguishability [6–8, 11, 16, 21, 22]; (ii) address only onion construction, without discussing or analyzing the routing algorithm [2, 9, 21–23]; (iii) consider routing only in the single run static network setting [3, 4, 24, 28, 32, 33]; or (iv) focus on lower bounds [4, 12, 17, 18].

2 Modeling the problem

Here, we introduce the setting used for our formal definitions in Section 3 and Section 5. Because our setting involves network churn, our model is more involved than previous models used for analogous definitions in the static setting. We base our treatment of churn on practical onion routing, namely Tor [19], which consists of thousands of routers and is used by millions of users. Tor relies on five to ten semi-trusted directory authorities to maintain up-to-date information on relay nodes and their availabilities and capabilities including network capacity. In the latest version (version 3) of Tor, routers periodically upload "router descriptors" that list their keys, capabilities, etc. to the directory authorities [1]. From these descriptors, the directory authorities update their view of the routers every 12 to 18 hours. Tor users and routers download "diffs" of the updated views from multiple directory authorities; these contain information on the currently available routers. The bulletins in our model are loosely modeled on this; like in Tor, at the start of every run, the parties obtain a global view of who's currently online. During a run, some parties may churn out; we allow the adversary to control who these parties are and when the churn happens since this corresponds to the most pessimistic scenario.

Notation. For a natural number n, $[n]$ is the set $\{1, \ldots, n\}$. For a set Set, we denote the cardinality of Set by $|\mathsf{Set}|$, and $\mathsf{item} \leftarrow\$ \mathsf{Set}$ is an item from Set chosen uniformly at random. If Dist is a probability distribution over Set, $\mathsf{item} \leftarrow \mathsf{Dist}$ is an item sampled from Set according to Dist. For an algorithm Algo, $\mathsf{output} \leftarrow \mathsf{Algo}(\mathsf{input})$ is the (possibly probabilistic) output from running Algo on input. A function $f(\lambda)$ of the security parameter λ is said to be *negligible* if it decays faster than any inverse polynomial in λ. An event occurs *with overwhelming probability* if its complement occurs with negligible probability.

System parameters. Let λ be the security parameter. Let N be the number of parties.

Time. We assume the synchronous setting and model time as passing in *rounds*, with some fixed number of rounds making up each larger *run*. Let $R_1, ..., R_L$ be a series of runs. Assume that L is bounded above by a polynomial in λ.

Parties. Let $P_1, ..., P_N$ be the N parties in our universe. Assume that N is bounded above by a polynomial in λ. Let $\mathsf{Bad} \subseteq \{P_1, ..., P_N\}$ be the set of corrupted parties. Corrupted parties are those that can be observed or controlled by the adversary, depending on the adversary's abilities, which we define later.

Churn bulletins. Let $B_1, ..., B_L$ be the *bulletins*, which accurately indicate which parties are online at the beginning of each run. More precisely, $B_i \subseteq \{P_1, ..., P_N\}$, and party P_j is online at the beginning of run R_i if and only if $P_j \in B_i$.

Churn schedule. Let $C_1, ..., C_L$ be the *churn schedule* for the runs. For each i, C_i is a set of party-round pairs: $C_i = \{(P_{i_1}, r_1), (P_{i_2}, r_2), ...\}$, where a pair (P_{i_j}, r_j) indicates that in run R_i, party P_{i_j} goes offline at the beginning of round r_j of that run. All parties in the list C_i must be online at the start of the run according to B_i. We allow parties to come online at the start of a run but not during a run. Thus the churn schedule specifies only which parties go offline and when. Since parties come online only at the start of a run, this will be specified in the bulletins rather than in the churn schedule.

Churn limit. Let $c(N)$, a function of N (e.g., $\frac{N}{2}$), be the *churn limit* [5]; that is, at most $c(N)$ parties can be offline at any point in time. We require that the number of offline parties specified by the bulletins and churn schedule does not exceed the churn limit $c(N)$. More precisely, for every i, $N - |B_i| + |C_i| \leq c(N)$ since $N - |B_i|$ parties are offline at the start of run R_i, and $|C_i|$ additional parties go offline during R_i.

Inputs. We represent an input for a run R_i as a vector $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, ..., \sigma_{i,N})$ where $\sigma_{i,j}$ is the input for party P_j. Each party's input is either a recipient-message pair (P_k, m) specifying that that party sends message m to party P_k, or it is $\perp$. An input of $\perp$ indicates that that party sends no information in that run (although that party can still send dummy messages in a protocol).

For run i, we say an input vector σ_i is *valid* if there exists some permutation $f : [B_i] \to [B_i]$ such that for each party $P_j \in B_i$, the input to P_j is $(f(P_j), m_j)$ for some message m_j. Furthermore, the input for each party $P_j \notin B_i$ is $\perp$. Our allowed inputs here are analogous to the "Simple I/O" setting in prior work (e.g., [3, 4, 32, 33]), adapted for churn.

For defining anonymity using a game-based approach, we allow the adversary to choose a pair of inputs for an onion routing protocol, and its goal is to determine, by running the protocol, which of these inputs the challenger chose. If the adversary can choose any two inputs without any constraints on its choices, then it can trivially win, for example, by choosing two inputs that differ on a corrupted party's input. Thus, the adversary is constrained to choose the two inputs from the same equivalence class [4]. We say two input vectors $\sigma = (\sigma_1, ..., \sigma_N)$ and $\sigma' = (\sigma'_1, ..., \sigma'_N)$ are *equivalent* w.r.t. the set of corrupted parties Bad if for all $P_j \in \mathsf{Bad}$, $\sigma_j = \sigma'_j$, and the number of honest messages for which P_j is the recipient in σ is the same as the number of honest messages for which P_j is the recipient in σ'. The content of these honest messages for which P_j is the recipient must also be the same in σ and σ'. We denote this equivalence $\sigma \equiv_{\mathsf{Bad}} \sigma'$.

Adversary model. The adversary can control the churn, observe the network traffic, and choose which parties to corrupt (if any). We define three classes of adversaries of varying capabilities: the network adversary, the passive adversary, and the active adversary. An adversary in any of these classes can observe all of the network traffic. In particular, it can observe the traffic on all communication links. The *network adversary* can control the churn, make these observations, and no more. The *passive adversary* can additionally corrupt a constant fraction of the parties and observe the computations and states of these parties. It cannot control these parties to deviate from the protocol. The *active adversary* can do all of the above and can also control the corrupted parties to do anything, including deviating from the protocol.

3 Onion encryption for churn

In this section, we formalize a generalization of duo onion encryption [21], where each onion has two candidate intermediary servers for each layer. As mentioned by Iwanik et al. [21] if the processing party can choose which candidate to send to next, then whenever possible, a corrupted party processing an onion can send to a corrupted candidate. If a fraction c of the participants are corrupted, the probability that at least one of a given list of k candidates is corrupted is $1 - c^k$.

As discussed in the introduction, we address this issue by introducing auxiliary parties, called *helpers*. The helpers for an onion O in a hop i are parties that are involved in some way in sending the peeled onion of O to its next intermediary server. This prevents a corrupted party from choosing any candidate it wishes for the next hop.

3.1 I/O syntax

Poly onion encryption is parameterized by the security parameter λ, the number of candidates κ, and the number of helpers ν and consists of algorithms ($\mathsf{KeyGen}, \mathsf{FormOnion}$) and protocol $\mathsf{ProcOnion}$, as follows:

- KeyGen takes as input the security parameter 1^λ, and a party name P_i. It outputs the public key pk_{P_i} and the secret key sk_{P_i} for P_i.
- $\mathsf{FormOnion}$ takes as input a message m; a run number R (recall that a run consists of a number of rounds; see Section 2); two ordered lists, $\mathcal{P}_1, \ldots, \mathcal{P}_\ell, \mathcal{P}_{\ell+1}$ and $\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell$ such that for all i, $|\mathcal{P}_i| = \kappa$ and $|\mathcal{Q}_i| = \nu$, and the public keys for all the parties on these lists. $\mathcal{P}_i$ is the ordered list of parties who are candidates for intermediaries for hop i. $\mathcal{Q}_i$ is the list of parties who will serve as helpers for hop i. The first party $P_{\ell+1,1}$ in $\mathcal{P}_{\ell+1}$ is the recipient.

  The output is the list of lists of onions $\mathcal{O}_1, \ldots, \mathcal{O}_{\ell+1}$. Each $\mathcal{O}_i$ corresponds to the i^{th} layer of this onion; each $\mathcal{O}_i$ consists of κ onions $O_{i,1}, \ldots O_{i,\kappa}$. An onion $O_{i,j}$ corresponds to the representation of the i^{th} layer of the onion that is suitable for processing by candidate $P_{i,j}$.
- $\mathsf{ProcOnion}$ is a protocol that $P_{i,j} \in \mathcal{P}_i$ can initiate on input $O_{i,j}$ and its secret key; the other participants in the protocol (if any) is the set of helpers $\mathcal{Q}_i$, each helper takes its own secret key as input. As a result of the protocol, $P_{i,j}$ obtains output $O_{i+1,j'} \in \mathcal{O}_{i+1}$ and its intended recipient $P_{i+1,j'} \in \mathcal{P}_{i+1}$; the helpers receive no output.

Remark 1. *We fix κ and ν for convenience; while in practice they may vary, fixing them does not lose much generality.*

Remark 2. *The list $\mathcal{P}_i = (P_{i,1}, \ldots, P_{i,\kappa})$ is ordered. For $j > 1$, $P_{i,j}$ is not supposed to serve as the intermediary for processing the onion unless for all $u < j$, $P_{i,u}$ is unavailable.*

Remark 3. *The candidate list $\mathcal{P}_{\ell+1}$ may seem superfluous: only the recipient $P_{\ell+1,1}$ is important. As we will see, requiring it as part of the input is helpful for preventing adversarial helpers from learning whether the onion to be processed has reached the end of the routing path, or not.*

Remark 4. *The recipient $P_{\ell+1,1}$ of the onion, upon receiving the onion $O_{\ell+1,1}$ should be able to process it, infer that he is the recipient, and obtain the original message m. An alternate candidate*

$P_{\ell+1,j}$, upon receiving the onion $O_{\ell+1,j}$ should be able to process it, infer that he is not the recipient, and output $\perp$. Correctness, defined in the next section, will ensure that this is the case.

3.2 Correctness

A standard onion encryption [2, 9, 22] is correct if having each intermediary peel it using the algorithm ProcOnion will get it to its destination and yield the original message. In poly onions, we have a set of candidates for each layer rather than a specific intermediary, and a set of helpers with which an intermediary can run ProcOnion in order to peel the onion; that alone makes correctness somewhat harder to pin down since now it is a protocol rather than an algorithm.

What makes it really complicated, however, is churn. A processing party may change its behavior based on whether the candidates for the next hop are online. In other words, processing an onion correctly depends on factors that cannot be accounted for at the time that the onion was formed.

We introduce a correctness predicate $\phi_{B,C}$ that corresponds to the bulletin B and churn schedule C. It takes as input the onion's candidate lists $\mathcal{P}$, the helper lists $\mathcal{Q}$, the pair of indices (i, j) where i is a hop in the routing path and j is the index of the j^{th} party $P_{i,j}$ in $\mathcal{P}_i$, a round r, and a number of rounds Δ. Δ should be an upper bound on the number of rounds required to process an onion. The correctness predicate $\phi_{B,C}(\mathcal{P}, \mathcal{Q}, (i, j), r, \Delta)$ accepts if $P_{i,j}$ and (a sufficient number of) helpers in $\mathcal{Q}_i$ are online at round r according to B and C. The definition of correctness is given with respect to this predicate (which, in turn, dictates how many helpers are sufficient).

In poly onion encryption, there is a set of onions rather than a single onion corresponding to each hop in the evolution. The path the onion will take through the network (i.e. which candidate will be picked for each hop) depends on which parties are online and which are corrupted. Recalling that $\mathcal{P}_{i+1}$ is a list of candidates in order of preference, O_i should not peel to an onion $O_{i+1,j}$ for party $P_{i+1,j} \in \mathcal{P}_{i+1}$ if there is an honest and online party $P_{i+1,k} \in \mathcal{P}_{i+1}$ where $k < j$. Note that if $P_{i+1,k}$ is online but corrupted, it may pretend to be offline, in which case we can allow O_i to peel to $O_{i+1,j}$. More formally:

Definition 1 (Correctness with respect to predicate ϕ). *Let $\Sigma = (\mathsf{KeyGen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$ be a poly onion encryption scheme, with ProcOnion taking at most Δ rounds to run. Let Bad be the set of corrupted parties. Let B be any bulletin. Let C be any churn schedule. Let m be any message. Let R be the current run number. Let $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_{\ell+1}) = ((P_{1,1}, \ldots, P_{1,\kappa}), \ldots, (P_{\ell+1,1}, \ldots, P_{\ell+1,\kappa}))$ be any list of $\ell + 1$ lists of κ candidates. Let $\mathcal{Q} = (\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell) = ((Q_{1,1}, \ldots, Q_{1,\kappa}), \ldots, (Q_{\ell,1}, \ldots, Q_{\ell,\kappa}))$ be any list of ℓ lists of ν helpers. Let $\mathsf{pk}_{\mathcal{P}\cup\mathcal{Q}}$ denote the public keys of the parties in $\mathcal{P} \cup \mathcal{Q}$.*

Let $\mathcal{O} = ((O_{1,1}, \ldots, O_{1,\kappa}), \ldots, (O_{\ell+1,1}, \ldots, O_{\ell+1,\kappa})) \leftarrow \mathsf{FormOnion}(m, R, \mathcal{P}, \mathcal{Q}, \mathsf{pk}_{\mathcal{P}\cup\mathcal{Q}})$ be an evolution of onions obtained from running FormOnion on the above parameters.

Σ is correct w.r.t. the predicate $\phi_{B,C}$ if for any candidate location (i, j), round r, and number of rounds Δ such that $\phi_{B,C}(\mathcal{P}, \mathcal{Q}, (i, j), r, \Delta) = 1$, the following items are satisfied:

i. Let $\mathcal{S} \subseteq \mathcal{P}_{i+1}$ be the following set of parties. If $\mathcal{P}_{i+1}$ contains an honest party that is online in rounds r through $r + \Delta$, $\mathcal{S}$ includes the first honest and online party P' in $\mathcal{P}_{i+1}$, along with any corrupted parties preceding P' in $\mathcal{P}_{i+1}$.

ii. When ProcOnion is initiated by an intermediary party $P_{i,j} \in \mathcal{P}_i$ in round r, and $P_{i,j}$ follows the protocol (i.e., if it is adversarial, then it can only be honest-but-curious), $P_{i,j}$'s output is $(P_{\mathsf{next}}, O_{\mathsf{next}})$ where $P_{\mathsf{next}} \in \mathcal{S} \cup \{\perp\}$. (The presence of $\perp$ on this list of parties means that it is possible that after the participants have processed the i^{th} layer of the onion, the adversary can drop this onion.)

iii. $O_{\mathsf{next}} = \bot$ *if* $P_{\mathsf{next}} = \bot$. *Otherwise,* $O_{\mathsf{next}} = O_{i+1,k}$ *is the onion layer for party* $P_{\mathsf{next}} = P_{i+1,k}$ *output by* FormOnion.

iv. When ProcOnion *is initiated by* $P_{\ell+1,1}$ *on input* $O_{\ell+1,1}$*, the output is* $(m, \bot)$*. When* ProcOnion *is initiated by* $P_{\ell+1,j}$ *on input* $O_{\ell+1,j}$*,* $j > 1$*, the output is* $(\bot, \bot)$*.*

Remark 5. *The evolution of an onion includes a representation of every layer of the onion, which is explicitly output by* FormOnion*. Implicitly, it also includes the innermost layer, i.e., the message that will ultimately be output by the recipient. Thus, we will sometimes think of an onion evolution with* ℓ *intermediaries as consisting of* $\ell+2$ *onion layers. For* $1 \le i \le \ell$*, an honest intermediary* $P_{i,j}$ *receives a representation* $O_{i,j}$ *of the* i^{th} *layer of* $\mathcal{O}$ *and, upon processing it, sends* $O_{i+1,j'}$ *to* $P_{i+1,j'}$*. If the recipient* $P_{\ell+1,1}$ *is online, it will receive the onion* $O_{\ell+1,1}$ *and, upon processing it, will output* $(m, \bot)$*. Sometimes, by* $O_{\ell+2,j}$ *we will denote* $(m, \bot)$*.*

3.3 Security

On a high level, an onion scheme is secure if an adversary cannot correlate an honest participant's incoming onions with its outgoing onions. For poly onions, this is captured via a security game, POSecurityGame described below.

One reason that this game is more complicated than the security game for regular onions is that the adversary controlling the helpers obtains additional information; what the adversary may learn also depends on the network churn. We introduce a security predicate ψ to capture whether or not a particular set of circumstances — who is processing an onion, at what round, with what helpers — dictates whether the adversary should not be able to determine a correlation.

More precisely, the security predicate $\psi_{B,C}$ is parameterized by a bulletin B and churn schedule C. Let $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_{\ell+1}) = ((P_{1,1}, \ldots, P_{1,\kappa}), \ldots, (P_{\ell+1,1}, \ldots, P_{\ell+1,\kappa}))$ be any list of $\ell+1$ lists of κ candidates. Let $\mathcal{Q} = (\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell) = ((Q_{1,1}, \ldots, Q_{1,\kappa}), \ldots, (Q_{\ell,1}, \ldots, Q_{\ell,\kappa}))$ be any list of ℓ lists of ν helpers. $\psi_{B,C}$ takes as input $\mathcal{P}$, $\mathcal{Q}$, a hop number h, a round r, and a number of rounds Δ.

For example, for regular onion routing, we would define ψ to be 1 if and only if the (only) candidate in hop h, P_h, is honest. Here, we don't need to refer to the bulletin or churn, since with regular onion routing, the adversary should not be able to peel an onion for honest P_h, regardless of whether or not P_h is online.

Consider another example, the original duo onion encryption [21] without helpers. Here, a processing party P_{h-1} in hop $h-1$ can choose which destination in $\mathcal{P}_h$ to send the onion to; there are no helpers verifying that this destination is the first candidate on the list $\mathcal{P}_h$ that is online. Here, we can define ψ to be 1 if and only if all parties in $\mathcal{P}_h$ are honest. If all parties in $\mathcal{P}_h$ are honest, the adversary should not be able to peel the onion since it does not know these honest parties' secret keys. On the other hand, if any party in $\mathcal{P}_h$ is corrupted, a corrupted P_{h-1} can choose to send to the corrupted party in $\mathcal{P}_h$, allowing the adversary to peel the onion in the following hop h. So the hop number h corresponds to the onion layer that shouldn't be "peelable" by the adversary.

POSecurityGame. The following game is between an adversary $\mathcal{A}$ and a challenger. It is parameterized by a security predicate $\psi_{B,C}(\mathcal{P}, \mathcal{Q}, h, r, \Delta)$, where B is a bulletin, C is a churn schedule, $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_{\ell+1})$ is a list of $\ell+1$ lists of candidates, $\mathcal{Q} = (\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell)$ is a list of ℓ lists of helpers, h is an index of a hop in the path, r is a round, and Δ is an upper bound on the number of rounds that ProcOnion takes to complete.

i. $\mathcal{A}$ receives the public keys for all parties.
ii. $\mathcal{A}$ chooses the set of corrupted parties Bad, the bulletin B, the churn schedule C, and the public keys for Bad. $\mathcal{A}$ sends all of these to the challenger.

iii. $\mathcal{A}$ can invoke the protocol ProcOnion in two ways, as follows.

Honestly initiated $\mathcal{A}$ sends to the challenger an onion O to be processed, an honest processing party P, and a round r_O in which the processing of O should begin. Next, the challenger acts on behalf of P as well as the honest helpers in the protocol ProcOnion initiated by P on input O, while $\mathcal{A}$ acts on behalf of the participants in Bad. Upon completing ProcOnion, the challenger reveals P's output (O', P') (if any) to $\mathcal{A}$.

Adversarially initiated $\mathcal{A}$ initiates ProcOnion on behalf of a participant $P \in \mathsf{Bad}$. Next, the challenger acts on behalf of the honest helpers in the protocol ProcOnion initiated by P on input O, while $\mathcal{A}$ acts on behalf of the participants in Bad (including P).

iv. $\mathcal{A}$ chooses the parameters for the challenge onion. It chooses a routing path length ℓ, a message m, a routing position $1 \leq h \leq \ell + 1$, a round r, a series of helper parties for each hop $(\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell)$, and a path consisting of a series of alternate destinations for each hop $(\mathcal{P}_1, \ldots, \mathcal{P}_{\ell+1})$. For the adversary's choices, it must hold that $\psi_{B,C}(\mathcal{P}, \mathcal{Q}, h, r, \Delta) = 1$.

v. The challenger samples a bit $b \leftarrow\!\!\$\ \{0, 1\}$. If $b = 0$, the challenger uses FormOnion to create an onion $\mathcal{O}^0$ exactly as specified by the routing path and helper parties. Let $\mathcal{O}_1^0$ be the list of outermost onion layers of this onion.

If $b = 1$, the challenger creates two lists of lists of onions. The challenger creates the first list of lists of onions $\mathcal{O}^1 = (\mathcal{O}_1^1, \ldots, \mathcal{O}_{h+1}^1)$ by running FormOnion with message $\perp$, candidates $(\mathcal{P}_1, \ldots, \mathcal{P}_{h+1})$, helpers $(\mathcal{Q}_1, \ldots, \mathcal{Q}_h)$, and those parties' public keys. The challenger creates the second list of lists of onions $\mathcal{O}' = (\mathcal{O}'_{h+1}, \ldots, \mathcal{O}'_{\ell+2})$ by running FormOnion with message m, candidates $(\mathcal{P}_{h+1}, \ldots, \mathcal{P}_{\ell+1})$, helpers $(\mathcal{Q}_{h+1}, \ldots, \mathcal{Q}_\ell)$, and those parties' public keys. Let $\mathcal{O}_1^1 = \mathcal{O}_1$ as formed above. (Recall that $\mathcal{O}_{\ell+2}^1$ consists of entries $O_{\ell+2,j} = (m, \perp)$ as explained in Remark 5.)

The challenger sends $\mathcal{O}_1^b$ to $\mathcal{A}$.

vi. $\mathcal{A}$ can again invoke the ProcOnion in two ways, with a slight modification if honestly initiated.

Honestly initiated $\mathcal{A}$ can direct honest participants to invoke ProcOnion as described in step iii. but with the following modification: it can only query an onion $O_j \in \mathcal{O}_h^b$ in round r. If $b = 0$ was chosen, the challenger follows the protocol ProcOnion.

If $b = 1$, and $\mathcal{A}$ directed $P_j \in \mathcal{P}_h$ to invoke ProcOnion on input $O_j \in \mathcal{O}_h^1$, the challenger begins by faithfully following the protocol on behalf of honest helpers and the honest P_j. Suppose that doing so produces output $O_{h+1,j'} \in \mathcal{O}_{h+1}^1$ and candidate $P_{h+1,j'} \in \mathcal{P}_{h+1}$. If $h \leq \ell$ (i.e., $P_j = P_{h+1,j}$ is an intermediary) or $(h, j) = (\ell + 1, 1)$ (i.e., $P_j = P_{h,j}$ is the onion's recipient), then instead of returning these to $\mathcal{A}$, the challenger switches the onion and returns $O_{h+1,j'} \in \mathcal{O}'_{h+1}$, $P_{h+1,j'}$ to $\mathcal{A}$.

Adversarially initiated Behavior is the same as defined in step iii.

vii. $\mathcal{A}$ submits a guess b' of b. See Figure 1 for a schematic of the poly onion security game.

We say $\mathcal{A}$ wins POSecurityGame if $b' = b$. A poly onion encryption scheme is secure if no efficient adversary can win POSecurityGame with non-negligible advantage; more formally:

Definition 2 (Poly Onion Security with respect to predicate ψ)**.** *We say a poly onion encryption scheme Σ is poly onion secure with respect to ψ against the class of adversaries $\mathbb{A}$ if for every adversary $\mathcal{A} \in \mathbb{A}$,* $\left|\Pr[\mathcal{A} \text{ wins } \mathsf{POSecurityGame}(\mathcal{A}, \Sigma, \lambda, \kappa, \nu, \psi_{\cdot,\cdot})] - \frac{1}{2}\right| = \mathsf{negl}(\lambda)$.

Remark 6. *During the* ProcOnion *protocol, the adversary may see additional information other than the oracle's output, depending on the adversary's capabilities. For example, the network adversary, passive adversary, and active adversary can see the traffic across all links during the protocol.*

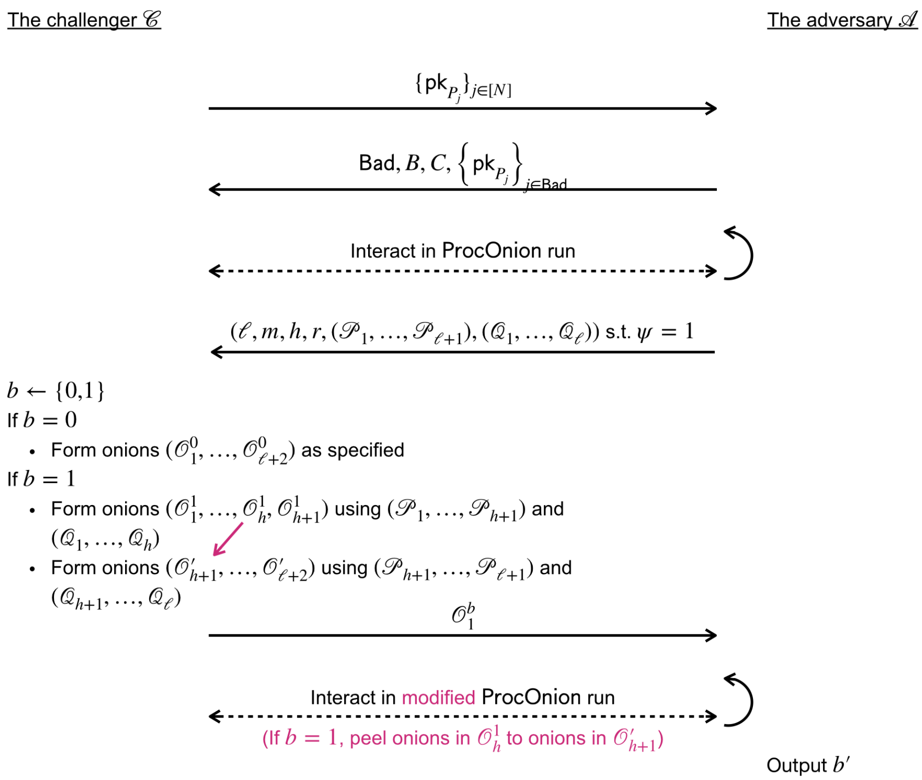

Figure 1: Schematic of the poly onion security game.

4 Our poly onion encryption scheme

In this section, we construct an instance of poly onion encryption, define correctness and security predicates, and prove its correctness and security with respect to these predicates.

Our construction, Poly Onion Encryption, has parameters κ (the number of candidates per hop), ν (the number of helpers per hop), α (the fraction of helpers needed to process an onion), and d (for bounding the length of the routing path). We construct our poly onion encryption scheme, Poly Onion Encryption, using the following cryptographic primitives: CCA secure public-key encryption with tags [13, 14], pseudo-random permutations (or block ciphers), a message authentication code (MAC) $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$, and a $(\alpha \cdot \nu, \nu)$ Secret Sharing scheme $(\mathsf{Share}, \mathsf{Recon})$. We denote public-key encryption and decryption as $\mathsf{Enc}_{\mathsf{pk}}(\cdot)$ and $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ where pk and sk are the public key and secret key, respectively. Following the work by Camenisch and Lysyanskaya [9] and Ando and Lysyanskaya [2], we will continue the tradition of using "$\{\cdot\}_k$" to denote evaluating a PRP in the forward direction under the symmetric key k, and "$\} \cdot \{_k$" to denote evaluating a PRP in the backward direction under k.

Throughout this section, we describe our construction for $\kappa = 2$ candidates per hop for ease of readability, although our construction generalizes to any $\kappa \in \mathbb{N}$. We explain how it generalizes in Section 4.5.

For every hop of the routing path, let P_i^+ denote the *preferred* candidate for the i^{th} hop (this is the sender's first choice), and let P_i^- denote the *alternate* candidate for the i^{th} hop (the second choice). The idea is that (at the i^{th} hop) the onion should be routed to P_i^+, unless P_i^+ is offline, in which case the onion can be routed to P_i^- instead. We sometimes refer to the party P_i for the i^{th} hop without specifying whether it is the preferred candidate or the alternate.

Forming an onion on input the message m, the candidate parties $\mathcal{P} = ((P_i^+, P_i^-))_{i\in[d]}$, the helpers (committee members) $\mathcal{Q} = (\mathcal{Q}_i)_{i\in[d]}$, and the public keys $\mathsf{pk}_{\mathcal{P}\cup\mathcal{Q}}$ of the candidates and helpers, produces a list of lists of onions, $((O_1^+, O_1^-), \ldots, (O_d^+, O_d^-)) \leftarrow \mathsf{FormOnion}(m, \mathcal{P}, \mathcal{Q}, \mathsf{pk}_{\mathcal{P}\cup\mathcal{Q}})$, where each O_i^+ is the onion to be processed by party P_i^+, and each O_i^- is the onion to be processed by P_i^-. If it is possible for the processing party P_i of an onion O_i to send an onion to the preferred next candidate P_{i+1}^+, P_i will produce an onion O_{i+1}^+ and send it to P_i^+; otherwise, P_i enlists the help of the committee members $\mathcal{Q}_i$ to produce the alternate onion O_{i+1}^- to send to the alternate candidate P_{i+1}^- instead. The point of this committee is to ensure that P_{i+1}^- can only process the onion if the preferred candidate P_{i+1}^+ is truly offline. Otherwise, a corrupted P_i could always choose to send to the corrupted party among P_{i+1}^+ and P_{i+1}^- if such a party exists, thereby significantly increasing the effective corruption rate.

4.1 Overview of Poly Onion Encryption

4.1.1 Anatomy of an onion

We describe at a high level the pertinent information contained in each onion $O_i = (K_i, H_i, U_i)$. A detailed description of how O_i is constructed is given in Section 4.2.

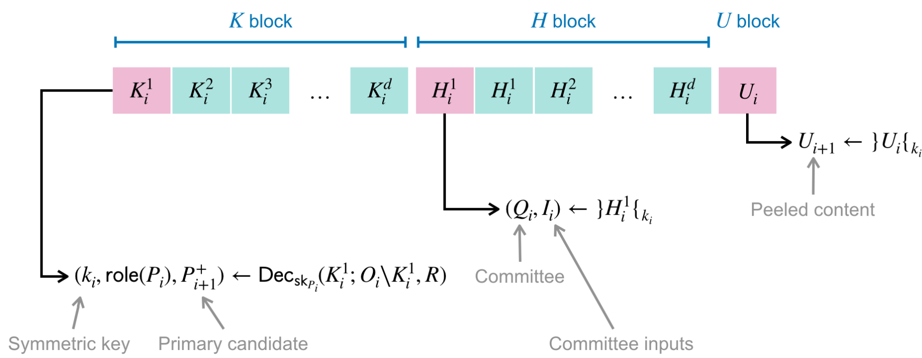

Figure 2: The structure of an onion O_i received by a processing party P_i.

- K_i contains d blocks, including a block for each hop j in the routing path. The first block K_i^1 is a ciphertext under the processing party's public key. It contains a key k_i, which will be used to decrypt the rest of the onion. K_i^1 also contains the role of the party (whether it is an intermediary or a recipient), and the identity P_{i+1}^+ of the preferred candidate for the next hop. The rest of the onion O_i (denoted $O_i \setminus K_i^1$), as well as the run/round number R, serves as the tag for this ciphertext; in other words, the ciphertext K_i^1 will not decrypt correctly unless the decryption occurs within the context of the correct onion O_i in run/round R:

$$(k_i, \mathrm{role}(P_i), P_{i+1}^+) \leftarrow \mathsf{Dec}_{\mathsf{sk}_{P_i}}(K_i^1; O_i \setminus K_i^1, R).$$

- H_i contains d blocks, including a block for each hop j in the routing path. Each H_i^j is encrypted using a block cipher with key k_i. The first block H_i^1 contains the identities of the committee $\mathcal{Q}_i$ and the set of inputs $I_i = \{I_{i,j}\}_{j\in[\nu]}$ for the committee to run the protocol.

$$(\mathcal{Q}_i, I_i) \leftarrow \}H_i^1\{_{k_i}.$$

- The input $I_{i,j}$ for the j^{th} committee member $Q_{i,j} \in \mathcal{Q}_i$ is $\mathsf{Enc}_{\mathsf{pk}_{Q_{i,j}}}(P_{i+1}^+, P_{i+1}^-, \sigma_{i,j}, T_{i,j}, R)$, where P_{i+1}^+ is the preferred candidate for the next hop, P_{i+1}^- is the alternate candidate for the next hop, $\sigma_{i,j}$ is $Q_{i,j}$'s share for reconstructing the alternate candidate's version of the onion, $T_{i,j}$ is the authentication tag for $\sigma_{i,j}$, and R is the run/round number when the $\mathsf{ProcOnion}$ protocol should take place. $\sigma_{i,j}$ verifies under the MAC with the tag $T_{i,j}$ and key k_i; that is, $\mathsf{Ver}_{k_i}(\sigma_{i,j}, T_{i,j}) =$ "accept."

- U_i contains the contents of the onion and is similar to the content in regular onion encryption. U_i is encrypted using a block cipher with key k_i.

4.1.2 Overview of processing an onion.

Let P_i be the processing party for onion $O_i = (K_i, H_i, U_i)$. Note that P_i can decrypt K_i^1 only if the onion wasn't modified en route; this is the purpose of using encryption with tags. P_i first decrypts K_i^1 with its secret key sk_{P_i}, to obtain the symmetric key k_i, learn its role role(P_i) (whether it is an intermediary for the onion or the recipient), and learn the identity of the preferred next destination P_{i+1}^+. The symmetric key k_i will allow P_i to decrypt the rest of the onion.

If the preferred next candidate P_{i+1}^+ is online, then P_i forms the "peeled" onion O_{i+1}^+ by decrypting the remaining blocks $K_i^2, \ldots, K_i^d, H_i^1, \ldots, H_i^d, U_i$ with k_i. P_i then shifts these blocks down so that, for example, $K_{i+1}^1 = \}K_i^2\{_{k_i}$. The last blocks of K_{i+1} and H_{i+1} are $K_{i+1}^d = \}11\ldots1\{_{k_{i+1}}$ and $H_{i+1}^d = \}00\ldots0\{_{k_{i+1}}$. This shifted, decrypted onion is $O_{i+1}^+ = (K_{i+1}^+, H_{i+1}, U_{i+1})$, the onion for P_{i+1}^+.

If the preferred next candidate P_{i+1}^+ is offline, P_i enlists the help of the committee $\mathcal{Q}_i$ to help peel the onion. It first decrypts H_1^i with k_i to obtain $\mathcal{Q}_i$ (the set of committee members) and I_i (the set of inputs for $\mathcal{Q}_i$). P_i initiates the protocol by sending each share $I_{i,j}$ to its corresponding committee member $Q_{i,j}$ in $\mathcal{Q}_i$. Each committee member $Q_{i,j}$ decrypts its input $I_{i,j}$ to obtain P_{i+1}^+, P_{i+1}^-, $\sigma_{i,j}$ (a sharing of the key block necessary to construct O_{i+1}^-), $T_{i,j}$ (the authentication tag for $\sigma_{i,j}$), and R (a run/round number). If R is not the current run/round, $Q_{i,j}$ aborts and outputs $\perp$. If $Q_{i,j}$ determines that P_{i+1}^+ is offline and P_{i+1}^- is online, it sends $\mathsf{Enc}_{\mathsf{pk}_{P_i}}(P_{i+1}^-, \sigma_{i,j}, T_{i,j})$ to P_i. Thus, if at least α fraction of the committee members are honest and online, and P_{i+1}^+ is offline and P_{i+1}^- is online, P_i will receive from the committee members, the identity of P_{i+1}^- and at least $\alpha|\mathcal{Q}_i|$ shares that verify using the set of tags T_i and the key k_i. P_i uses these shares to reconstruct the alternate first key block $(K_{i+1}^1)^-$. P_i now processes the rest of the onion as in the case where P_{i+1}^+ is online, decrypting the other blocks with k_i and shifting them down, then again forming K_{i+1}^d and H_{i+1}^d as encryptions of $00\ldots0$ and $11\ldots1$ respectively. It then replaces the first key block $(K_{i+1}^1)^+$ of K_{i+1}^+ with the reconstructed key block $(K_{i+1}^1)^-$ to obtain K_{i+1}^-. The resulting peeled onion is the alternate onion $O_{i+1}^- = (K_{i+1}^-, H_{i+1}, U_{i+1})$.

P_i can then relay the peeled version of the onion to its corresponding candidate, who holds the secret key required to peel it. We describe our construction in more detail in the following sections.

4.2 Forming an onion

We need to construct the onion such that each component contains the information previously described. Our onions have d K-blocks, d H-blocks, and a single content block U. We assume that the length ℓ of the routing path always satisfies $\ell + 1 < d$, so there are enough blocks to complete the routing path.

We first construct the onions $\mathcal{O}_\ell$ for the recipient, then describe a subroutine called "wrapping" an onion that allows us to form the entire sequence of onions. We are given the onion parameters: a message m, candidate lists $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$, and committees $\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell$. An onion has block length d; that is, each onion has d K-blocks and d H-blocks.

We begin by sampling the key k_ℓ using the key generation algorithm for the block cipher. For every $i \in \{2, \ldots, d\}$, we let $K_\ell^i = \}11\ldots1\{_{k_\ell}$. For every $i \in \{2, \ldots, d\}$, we let $H_\ell^i = \}00\ldots0\{_{k_\ell}$. Let P_ℓ^+ denote the primary candidate for hop ℓ; let P_ℓ^- denote the alternate candidate for hop ℓ. Let $\sigma_\ell \leftarrow \mathsf{Share}(00\ldots0)$, and let each tag $T_{\ell,j} \leftarrow \mathsf{Tag}_{k_\ell}(\sigma_{\ell,j})$. Let each input $I_{\ell,j} \leftarrow \mathsf{Enc}_{\mathsf{pk}_{\mathcal{Q}_{\ell,j}}}(P_{\ell+1}^+, P_{\ell+1}^-, \sigma_{\ell,j}, T_{\ell,j}, R_{\ell+1})$, where $R_{\ell+1}$ includes the current run number and the hop number $\ell + 1$. Let $H_\ell^1 = \{\mathcal{Q}_\ell, I_\ell\}_{k_\ell}$, and $H_\ell = (H_\ell^1, \ldots, H_\ell^d)$. Let $U_\ell = \{m\}_{k_\ell}$. Let $(K_\ell^1)^+ = \mathsf{Enc}_{\mathsf{pk}_{P_\ell^+}}(k_\ell, \mathrm{role}(P_\ell^+), P_{\ell+1}^+)$ where $\mathrm{role}(P_\ell^+)$ is recipient, and let $(K_\ell^1)^- = \mathsf{Enc}_{\mathsf{pk}_{P_\ell^-}}(k_\ell, \mathrm{role}(P_\ell^-), P_{\ell+1}^+)$ where $\mathrm{role}(P_\ell^-)$ is recipient. Both $(K_\ell^1)^+$ and $(K_\ell^1)^-$ are encrypted with tag $(K_\ell^2, K_\ell^3, \ldots, K_\ell^d, H_\ell, U_\ell), R_\ell$, where R_ℓ includes the current run number and the hop number ℓ.

Let $K_\ell^+ = ((K_\ell^1)^+, \ldots, K_\ell^d)$, and let $K_\ell^- = ((K_\ell^1)^-, \ldots, K_\ell^d)$. The innermost set of onion layers is $\mathcal{O}_\ell = (O_\ell^+, O_\ell^-)$, where $O_\ell^+ = (K_\ell^+, H_\ell, U_\ell)$ and $O_\ell^- = (K_\ell^-, H_\ell, U_\ell)$.

We then wrap O_ℓ^+ using the parameters $\mathcal{P}_{\ell-1}, \mathcal{P}_\ell, \mathcal{Q}_{\ell-1}$ to obtain $\mathcal{O}_{\ell-1} = (O_{\ell-1}^+, O_{\ell-1}^-)$. We repeatedly wrap O_i^+ with parameters $\mathcal{P}_i, \mathcal{P}_{i+1}, \mathcal{Q}_i$ to obtain $\mathcal{O}_{i-1} = (O_{i-1}^+, O_{i-1}^-)$, until we have $\mathcal{O}_1$, the outermost set of layers for the onion. $\mathsf{FormOnion}$ returns the list of onion layers for each hop:

$$(O_1^+, O_1^-), \ldots, (O_\ell^+, O_\ell^-)$$

4.2.1 Wrapping an onion

Given an onion O_{i+1}^+, we can "wrap" it in a layer of encryption to obtain another list of onions $\mathcal{O}_i$, each of which peels to O_{i+1}^+ or O_{i+1}^-. Formally, we are given the onion O_{i+1}^+, where

$$O_{i+1}^+ = (K_{i+1}^+, H_{i+1}, U_{i+1})$$

and we are given the relevant wrapping parameters for round i: the current candidates $\mathcal{P}_i$, the next candidates $\mathcal{P}_{i+1}$, and the committee $\mathcal{Q}_i$. We sample a key k_i for the block cipher using its key generation algorithm. We then form the set of shares $\sigma_i \leftarrow \mathsf{Share}((K_{i+1}^1)^+)$ for the committee members in $\mathcal{Q}_i$. Using the shares, we form the set of tags T_i, where $T_{i,j} \leftarrow \mathsf{Tag}_{k_i}(\sigma_{i,j})$. For each committee member $Q_{i,j}$, we create an input $I_{i,j}$, where R_i includes the current run number and hop number i:

$$I_{i,j} \leftarrow \mathsf{Enc}_{\mathsf{pk}_{Q_{i,j}}}(P_{i+1}^+, P_{i+1}^-, \sigma_{i,j}, T_{i,j}, R_i)$$

We form the first H-block in O_i^+ and O_i^- as follows:

$$H_i^1 = \{\mathcal{Q}_i, I_i\}_{k_i}$$

We form the rest of O_i^+ and O_i^- by modifying ("wrapping") blocks of O_{i+1}^+.

$$K_i^j = \{K_{i+1}^{j-1}\}_{k_i}, 2 \leq j \leq d$$
$$H_i^j = \{H_{i+1}^{j-1}\}_{k_i}, 2 \leq j \leq d$$
$$U_i = \{U_{i+1}\}_{k_i}$$

We form the blocks $(K_i^1)^+$ and $(K_i^1)^-$, encrypting them each with the tag $(K_i^2, K_i^3, \ldots, K_i^d, H_i, U_i, R_i)$:

$$(K_i^1)^+ \leftarrow \mathsf{Enc}_{\mathsf{pk}_{P_i^+}}(k_i, \mathrm{role}(P_i^+), P_{i+1}^+)$$

$$(K_i^1)^- \leftarrow \mathsf{Enc}_{\mathsf{pk}_{P_i^-}}(k_i, \mathrm{role}(P_i^-), P_{i+1}^+)$$

We let $K_i^+ = ((K_i^1)^+, K_i^2, \ldots, K_i^d)$ and $O_i^+ = (K_i^+, H_i, U_i)$. We let $K_i^- = ((K_i^1)^-, K_i^2, \ldots, K_i^d)$ and $O_i^- = (K_i^-, H_i, U_i)$.

4.3 Processing an onion

When a processing party P_i obtains an onion $O_i = (K_i, H_i, U_i)$ in run/round R, P_i first uses its secret key sk_{P_i} and the tags $O_i \setminus K_i^1, R$ to decrypt the first block K_i^1 of K_i and obtain its role, a key k_i, and the identity P_{i+1}^+ of the preferred candidate for the next hop. If P_i's role is recipient, it decrypts U_i using k_i to obtain the message m and we are done. Otherwise, if P_i is an intermediary, it must peel O_i. If P_{i+1}^+ is online, P_i need not involve the committee. It peels O_i using the symmetric key k_i, decrypting each block of the K, H, and U sections, and shifting the K and H blocks so that the first blocks contain the relevant information for the next hop:

$$
\begin{aligned}
(k_i, \mathrm{role}(P_i), P_{i+1}^+) &\leftarrow \mathsf{Dec}_{\mathsf{sk}_{P_i}}(K_i^1; O_i \setminus K_i^1, R) \\
H_{i+1}^j &= \}H_i^{j+1}\{_{k_i} \text{ for } 1 \le j \le d-1 \\
H_{i+1}^d &= \}00\ldots0\{_{k_i} \\
K_{i+1}^j &= \}K_i^{j+1}\{_{k_i} \text{ for } 1 \le j \le d-1 \\
K_{i+1}^d &= \}11\ldots1\{_{k_i} \\
U_{i+1} &= \}U_i\{_{k_i}
\end{aligned}
$$

Let $(K_{i+1})^+ = (K_{i+1}^1, \ldots, K_{i+1}^d)$, and let $H_{i+1} = (H_{i+1}^1, \ldots, H_{i+1}^d)$. In the case that P_i observes that P_{i+1}^+ is online, the processed version of O_i is simply $O_{i+1}^+ = ((K_{i+1})^+, H_{i+1}, U_{i+1})$.

If P_i observes that P_{i+1}^+ is offline, P_i enlists the help of the committee $\mathcal{Q}_i$ to construct O_{i+1}^-. It uses k_i to decrypt H_i^1 and obtain the identities of the committee members $\mathcal{Q}_i$ and the set of committee members' inputs I_i.

$$\mathcal{Q}_i, I_i = \}H_i^1\{_{k_i}$$

P_i initiates the $\mathsf{ProcOnion}$ protocol by sending input $I_{i,j}$ to party $Q_{i,j} \in \mathcal{Q}_i$ for $j \in [|\mathcal{Q}_i|]$. Each committee member $Q_{i,j}$, upon receiving its input $I_{i,j}$, decrypts $I_{i,j}$ with its secret key. It obtains the identities of the candidates P_{i+1}^+ and P_{i+1}^- for the next hop, its share $\sigma_{i,j}$, the authentication tag $T_{i,j}$, and a run/round number. If the run/round number is not the current run/round, $Q_{i,j}$ aborts and sends $\mathsf{Enc}_{\mathsf{pk}_{P_i}}(\perp, \perp)$ to P_i. If P_{i+1}^+ is offline and P_{i+1}^- is online, $Q_{i,j}$ sends $\mathsf{Enc}_{\mathsf{pk}_{P_i}}(P_{i+1}^-, \sigma_{i,j}, T_{i,j})$ to P_i. If P_{i+1}^+ is online or both P_{i+1}^+ and P_{i+1}^- are offline, $Q_{i,j}$ sends $\mathsf{Enc}_{\mathsf{pk}_{P_i}}(\perp, \perp)$ to P_i.

P_i decrypts each output with its secret key. For each share $\sigma_{i,j}$ that P_i receives, it checks that $\mathsf{Ver}_{k_i}(\sigma_{i,j}, T_{i,j}) =$ "accept". If P_i receives at least $\alpha \cdot \nu$ shares that verify, P_i can reconstruct the block $(K_{i+1}^1)^-$, which is a version of the first K block for O_{i+1}^- that can be decrypted using the secret key of alternate candidate P_{i+1}^-. If the majority of the committee is honest and online, the protocol will terminate with P_i receiving the name of the party P_{i+1}^- and $(K_{i+1}^1)^-$. P_i then replaces the first block of $(K_{i+1})^+$ with $(K_{i+1}^1)^-$, yielding $(K_{i+1})^- = ((K_{i+1}^1)^-, K_{i+1}^2, \ldots, K_{i+1}^d)$. P_i then sends onion $O_{i+1}^- = ((K_{i+1}^1)^-, H_{i+1}, U_{i+1})$ to P_{i+1}^-.

If P_i does not receive enough verified shares to reconstruct $(K_{i+1})^-$, O_i is dropped. Otherwise, if processing is successful, P_i relays the peeled onion O_{i+1}^+ or O_{i+1}^- to the first of P_{i+1}^+ and P_{i+1}^- that is online.

4.4 Analysis of Poly Onion Encryption

Here, we analyze Poly Onion Encryption for $\kappa = 2$.

Correctness. We define the predicate function $\phi_{B,C,\alpha}^{\mathsf{poly}}(\mathcal{P}, \mathcal{Q}, (i,j), r, \Delta)$ to be 1 when $P_{i,j}$ is honest and online in rounds r through $r + \Delta$, and fewer than $\alpha \cdot \nu$ of the parties in $\mathcal{Q}_i$ are corrupted.

Poly Onion Encryption is correct with respect to $\phi^{\mathsf{poly}}_{B,C,\alpha}(\mathcal{P}, \mathcal{Q}, (i,j), r, \Delta)$. Suppose $P_{i,j}$ is honest and initiates the ProcOnion protocol on an onion $O_{i,j}$ in round r. We break the scenario into the following cases:

P^+_{i+1} **honest and online.** $P_{i,j}$ does not need the committee to process $O_{i,j}$. Since $P_{i,j}$ is honest, it will output (P^+_{i+1}, O^+_{i+1}) as prescribed by correctness.

P^+_{i+1} **honest and offline.** $P_{i,j}$ will see that P^+_{i+1} is offline and will enlist the help of the committee. The committee protocol returns either $\perp$ or the key block for O^-_{i+1}. Thus, $P_{i,j}$ will either output (P^-_{i+1}, O^-_{i+1}) or $\perp$.

P^+_{i+1} **corrupted.** Depending on whether P^+_{i+1} behaves as if it is online, $P_{i,j}$ may output $(P^+_{i+1}, O^+_{i+1}), (P^-_{i+1}, O^-_{i+1})$, or $\perp$.

The output in each of these cases is consistent with the definition of correctness in Definition 1.

Security. Let $\psi^{\mathsf{poly}}_{B,C,\alpha}(\mathcal{P}, \mathcal{Q}, h, r, \Delta)$ be the predicate function that returns 1 if and only if the following both hold:

i. No corrupted party precedes the first honest party in $\mathcal{P}_h$ that is online in all rounds r through $r + \Delta$.

ii. Fewer than $\alpha \cdot \nu$ parties in $\mathcal{Q}_{h-1}$ are corrupted.

Recall that ν is the committee size and α is the number of committee members' shares required to reconstruct the onion for the alternate candidate. By the above definition of $\psi^{\mathsf{poly}}_{B,C,\alpha}$, if the first candidate in $\mathcal{P}_{h+1}$ is honest and online, and fewer than $\alpha \cdot \nu$ members in $\mathcal{Q}_h$ are corrupted, the adversary cannot win the security game with non-negligible advantage, i.e., the onion mixes in hop $h + 1$. As long as enough parties in our universe are honest, and committees are chosen randomly, we can increase the committee size to boost the probability that fewer than $\alpha \cdot \nu$ members in $\mathcal{Q}_h$ are corrupted; we discuss this further in Section 6. Given that fewer than $\alpha \cdot \nu$ members of $\mathcal{Q}_h$ are corrupted, the onion mixes in hop $h + 1$ if the first party in $\mathcal{P}_{h+1}$ is honest and online. We show later in Section 6 that $\psi^{\mathsf{poly}}_{B,C,\alpha}$ is indeed satisfied with high enough probability to provide anonymity.

Theorem 1 (Security of construction)**.** *Poly Onion Encryption is poly onion secure with respect to the security predicate* $\psi^{\mathsf{poly}}_{B,C,\alpha}$ *for* $0 < \alpha \leq 1$ *and* $\nu \geq \frac{1}{\alpha}$ *assuming that all of the underlying standard primitives exist.*

We prove that the scheme is secure using a hybrid argument that is similar to the security proof of shallot encryption by Ando and Lysyanskaya [2]. We give a proof sketch below and provide thee full proof in appendix A.

Proof sketch. Let $\mathsf{Experiment}^0$ be the same as running the security game with $b = 0$; this is when the challenger creates the challenge onion as usual. Let $\mathsf{Experiment}^1$ be the same as running the security game with $b = 1$; this is when the challenger creates two unrelated sets of onion layers $\mathcal{O}$ and $\mathcal{O}'$, and the onion $O \in \mathcal{O}$ peels to $O' \in \mathcal{O}'$ at the chosen server.

We construct the following hybrids that act as stepping stones from $\mathsf{Experiment}^0$ to $\mathsf{Experiment}^1$. Let $i = h - 1$. The hybrids involve changing the onion layers $\mathcal{O}_{i+1}$. In all of the hybrids, the ProcOnion oracle behaves as if $b = 1$ in POSecurityGame. That is, when an onion $O_j \in \mathcal{O}_{i+1}$ is queried, it returns the onion in $\mathcal{O}_{i+2}$ corresponding to the appropriate candidate. This behavior is consistent with $b = 0$ in POSecurityGame for $\mathsf{Experiment}^0$ and with $b = 1$ in POSecurityGame for $\mathsf{Experiment}^1$:

$\mathsf{Experiment}^0$: security game with $b = 0$.

$\updownarrow$ These are identically distributed.

Hybrid^1: since onions are layered encryption objects, we form challenge onion by first forming O^+_{i+2} and then "wrapping" it in more layers of encryption to get $\mathcal{O}_1$. We formally define wrapping in Section 4.2.1.

$\updownarrow$ Indistinguishable by security of public key encryption.

Hybrid^2: same as Hybrid^1, except change the oracle so that in step 6 of $\mathsf{POSecurityGame}$, if it is queried with $(O_{i+1}^-)'$ to be processed by P_{i+1}^-, it instead runs $\mathsf{ProcOnion}$ with O_{i+1}^-.

$\updownarrow$ Indistinguishable by security of secret sharing/public key encryption.

Hybrid^3: same as Hybrid^2, except in block H_i^1 of $\mathcal{O}_i$, change the share of every member of committee $\mathcal{Q}_i$ to a share of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(00\ldots0)$ instead of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \mathrm{role}(P_{i+1}^-), P_{i+2}^+)$.

$\updownarrow$ Indistinguishable by security of public key encryption.

Hybrid^4: same as Hybrid^3, except in the key block K_{i+1}^1 of $\mathcal{O}_{i+1}$, change k_{i+1} to $00\ldots0$.

$\updownarrow$ Indistinguishable by security of the block cipher.

Hybrid^5: same as Hybrid^4, except change $\mathcal{O}_{i+1}$ from a wrapping of O_{i+2}^+ to the output for hop $(i+1)$ of $\mathsf{FormOnion}$ on the first segment of the routing path, up to P_{i+1}.

$\updownarrow$ Indistinguishable by security of public key encryption.

Hybrid^6: same as Hybrid^5, except in the key block K_{i+1}^1 of $\mathcal{O}_{i+1}$, change the key back from $00\ldots0$ to k_{i+1}, and change the role of P_{i+1} from intermediary to recipient.

$\updownarrow$ Indistinguishable by security of secret sharing/public key encryption.

Hybrid^7: same as Hybrid^6, except in block H_i^1 of $\mathcal{O}_i$, change all committee members' shares back from shares of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(00\ldots0)$ to shares of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \mathrm{role}(P_{i+1}^-), P_{i+2}^+)$

$\updownarrow$ Indistinguishable by security of public key encryption.

Hybrid^8: same as Hybrid^7, except change the oracle so that it no longer treats $(O_{i+1}^-)'$ specially.

$\updownarrow$ These are identically distributed.

$\mathsf{Experiment}^1$: security game with $b = 1$. □

4.5 Generalizing to more than two candidate processing parties

We remark that this construction can be generalized for any number of candidates κ. That is, every onion has κ candidate processing parties per hop. We can do so by modifying the committee members' inputs so that each input $I_{i,j}$ contains the full list of candidates rather than just P_{i+1}^+ and P_{i+1}^-. We also include $\kappa - 1$ shares $\sigma_{i,j}^2, \ldots, \sigma_{i,j}^\kappa$ in $I_{i,j}$ instead of just $\sigma_{i,j}$. Each share $\sigma_{i,j}^c$ is used to construct the version of the onion for candidate $P_{i+1,c} \in \mathcal{P}_{i+1}$. The processing party knows from the committee members' responses which candidate each committee member votes for. If enough committee members vote for one of the candidates, the processing party can reconstruct that candidate's version of the onion. Correctness and security still hold with respect to the same predicates $\phi_{B,C,\alpha}^{\mathsf{poly}}$ and $\psi_{B,C,\alpha}^{\mathsf{poly}}$ defined in Section 4.4. The proof of correctness is given below, and the proof of security is given in appendix A.

4.6 Correctness of Poly Onion Encryption with multiple candidates

When Poly Onion Encryption is implemented with multiple candidates, correctness still holds with respect to the same predicate $\phi_{B,C,\alpha}^{\mathsf{poly}}$ from Section 4.4. We again consider honest $P_{i,j}$ processing an onion $O_{i,j}$ and break this down into cases. The first three cases address when an honest and online party precedes all corrupted parties in $\mathcal{P}$. Again, let P_{i+1}^+ denote the preferred candidate for hop $i+1$. Let P_{i+1}^* denote the first honest party in $\mathcal{P}$ that is online in rounds r through $r+\Delta$.

P_{i+1}^+ online and honest. If P_{i+1}^+ is online and honest, honestly behaving $P_{i,j}$ will process the onion to produce (O_{i+1}^+, P_{i+1}^+) without the help of the committee.

$P_{i+1}^* \neq P_{i+1}^+$ precedes all corrupted parties in $\mathcal{P}$. If a different party P_{i+1}^* is the first online and honest party and is preceded by offline honest parties, $P_{i,j}$ will enlist the help of the committee. The honest and online committee members will see that P_{i+1}^* is the first online party and will return the corresponding shares. Since at least $\alpha \cdot \nu$ committee members are honest and online, $P_{i,j}$ will receive a sufficient number of shares to reconstruct (P^*_{i+1}, O^*_{i+1}).

All parties in $\mathcal{P}_{i+1}$ are offline and honest. If all parties in $\mathcal{P}_{i+1}$ are offline and honest, $P_{i,j}$ will not receive enough shares and will output $\bot$.

A corrupted party precedes P^*_{i+1}. Corrupted parties can choose to behave as if they are online or offline. In doing so, a corrupted party can cause the honest committee members to return shares corresponding to it, the honest party after it in $\mathcal{P}$, or possible no share at all (if all parties in $\mathcal{P}$ are corrupted or offline). Thus, the output onion could be for P^*_{i+1}, for any corrupted party preceding P^*_{i+1}, or $\bot$.

5 Anonymity in the setting with churn

So far we have explored new onion encryption techniques for handling network churn, defining poly onion encryption, and constructing a scheme that satisfies poly onion security. In this section, we turn our attention to the problem of how to route onions such as those constructed using Poly Onion Encryption through a dynamic network to achieve anonymity. To begin with, we must first formally define what it means for an onion routing protocol to be anonymous in a setting with network churn. Our new definitions of anonymity, including multi-run anonymity, are provided in Section 5.1.

To establish that our proposed multi-run anonymity definition is a usable notion, we must also show that it is achievable. In Section 5.3, we prove a general theorem (Theorem 2) that states that for a class of onion routing protocols, which we call "simulatable" protocols, single-run anonymity is equivalent to multi-run anonymity. An implication of this is that all previously known simulatable protocols that are single-run anonymous are also multi-run anonymous. These include Π_p [3]. However, these new multi-run results are for the static setting, without network churn. In Section 6, we prove (again relying on Theorem 2) that Π_p can achieve multi-run anonymity in the presence of churn. Our formal definition of the class of simulatable onion routing protocols is provided in Section 5.2.

5.1 Definitions of anonymity

Here, we define what it means for an onion routing protocol to achieve multi-run anonymity. First, we define an anonymity game, $\mathsf{StrongAnonGame}$, which we then use in the formal definition of multi-run anonymity (Definition 3).

$\mathsf{StrongAnonGame}(\mathcal{A}, \Pi, L, \lambda)$ is parameterized by the adversary $\mathcal{A}$, the onion routing protocol Π, the number of runs L, and the security parameter λ. The game proceeds in three phases: (i) the setup phase where $\mathcal{A}$ has access to the oracle for responding to queries for processing onions on behalf of honest parties, (ii) the challenge phase where $\mathcal{A}$ and the challenger run the protocol Π, and (iii) the final phase where $\mathcal{A}$ again has access to the oracle.

During *setup*, the adversary $\mathcal{A}$ first picks the set of corrupted parties Bad and sends Bad to the challenger. The challenger generates the keys for the honest parties according to Π and sends only the public portion of these keys to $\mathcal{A}$. $\mathcal{A}$ sends the corrupt parties' public keys to the challenger. $\mathcal{A}$ can now submit $\mathsf{ProcOnion}$ queries to the the challenger. For each $\mathsf{ProcOnion}$ query, $\mathcal{A}$ submits a bulletin B and a churn schedule C such that the number of parties ever offline is bounded above by the churn limit $c(N)$, an onion O, an honest processing party P for peeling O, and a round number r. The challenger interacts with $\mathcal{A}$ to run the $\mathsf{ProcOnion}$ protocol on O starting in round r, with the challenger acting on behalf of the honest parties following the protocol and $\mathcal{A}$ controlling the behavior of the corrupted parties.

In the *challenge phase*, $\mathcal{A}$ and the challenger run the protocol L times. To begin with, the challenger picks the challenge bit $b \in \{0, 1\}$. For each of the L runs, $\mathcal{A}$ and the challenger repeat the same procedure. In run i, $\mathcal{A}$ picks a bulletin B_i and a churn schedule C_i with at most $c(N)$ parties offline during that run. $\mathcal{A}$ also picks input vectors σ_0^i and σ_1^i that are both valid with respect to B_i, i.e., $\sigma_0^i \equiv_{\mathsf{Bad}} \sigma_1^i$. $\mathcal{A}$ sends B_i, C_i, σ_0^i, and σ_1^i to the challenger. $\mathcal{A}$ and the challenger interact in a protocol run of σ_b^i with online parties specified by bulletin B_i and churn schedule C_i, and with the challenger acting as the honest parties, and $\mathcal{A}$ acting as the corrupt parties.

After the challenge phase, in the *final phase*, $\mathcal{A}$ can again interact with the challenger by submitting $\mathsf{ProcOnion}$ queries, with the additional restriction that $\mathcal{A}$ cannot ask about onions formed by honest parties during the challenge phase. That is, $\mathcal{A}$ picks a bulletin B and churn schedule C (such that the number of offline parties is at most $c(N)$), an onion O (not observed during the challenge phase), and a processing party P. Finally, $\mathcal{A}$ outputs a guess b' of the challenge bit b. We say $\mathcal{A}$ wins $\mathsf{StrongAnonGame}(\mathcal{A}, \Pi, L, \lambda)$ if its guess b' is equal to b. See Figure 3 for a schematic of the strong anonymity game.

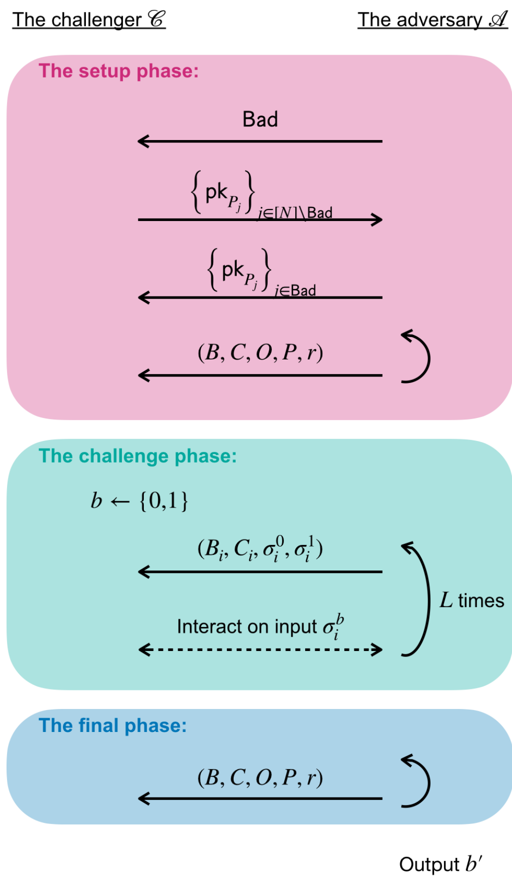

Figure 3: Schematic of the strong anonymity game.

We now define strong anonymity using the game $\mathsf{StrongAnonGame}$, along with several variants of this definition.

Definition 3 (Strong Anonymity)**.** *An onion routing protocol Π with security parameter λ is L-strongly anonymous against the class of adversaries $\mathbb{A}$ if for every adversary $\mathcal{A} \in \mathbb{A}$, $\left|\Pr[\mathcal{A}\ wins\ \mathsf{StrongAnonGame}(\mathcal{A}, \Pi, L, \lambda)] - \frac{1}{2}\right| = \mathsf{negl}(\lambda)$.*

Multi-run vs. single-run. We say a protocol Π is multi-run anonymous if it is anonymous for polynomially bounded $L > 1$ in the above definition. A protocol Π is single-run anonymous if it is anonymous for $L = 1$.

Strong vs. weak. We say a protocol Π is weakly anonymous if it satisfies the analogous definition for a modified anonymity game, where the adversary does not have oracle access to the ProcOnion queries. We say a protocol Π is strongly anonymous if it satisfies the definition using StrongAnonGame.

Adaptive vs. non-adaptive. In StrongAnonGame, the adversary is adaptive in that it can choose the bulletin, the schedule, and the inputs before each run based on prior history. We also define a weaker non-adaptive anonymity definition, in which the adversary must choose all inputs, churn schedules, and bulletins before observing any protocol runs. Using our terminology above, the standard anonymity definition in prior papers (e.g., [2,3,24,32,33]) is weak single-run non-adaptive anonymity in our new terms.

5.2 Simulatable onion routing protocols

Here, we formally define the class of simulatable onion routing protocols. As we will show in Section 5.3, simulatability is a property that can reduce multi-run anonymity to single-run anonymity. The idea is that if a simulatable onion routing protocol is single-run anonymous, then we can prove that it is also multi-run anonymous via a sequence of reductions that "simulate" extraneous runs for an adversary that expects to interact in multiple runs. (See our proof of Theorem 2 in Section 5.3.)

Thus, what we mean by "simulatable" is that the reduction should be able to recreate what the honest parties do in a run, using only information that it has access to – namely, the public keys of all the parties, the bulletin, the churn schedule, the run number, and the inputs for the honest parties. Consider the following two settings: (i) the real setting, in which the challenger interacts with the adversary by following the protocol and (ii) the ideal setting, in which the challenger interacts with the adversary by using the algorithm GenOnions that generates (from just the public parameters and the honest parties' inputs) all possible onions that the honest parties might send out during the run and the algorithm ScheduleProcOnions that determines (from just the honest parties' message buffers) if/when these onions are processed. An onion routing protocol is simulatable if no (efficient) adversary can tell whether it is interacting in the real setting or the ideal one except with negligible advantage. We define these concepts more concretely below.

The real setting. $\mathsf{RealGame}(\mathcal{A}, \Pi, \lambda)$ is parametrized by the adversary $\mathcal{A}$, the onion routing protocol Π with onion encryption scheme ($\mathsf{KeyGen}, \mathsf{FormOnion}, \mathsf{ProcOnion}$), and the security parameter λ.

The game proceeds as follows. First, the adversary $\mathcal{A}$ chooses the adversarial parties Bad, the bulletin B, the churn schedule C, the run number R, and the keys for the parties in Bad. The public portions of these keys are relayed to the challenger. The challenger generates the keys for the honest parties by running KeyGen and relays the public portion of these keys to $\mathcal{A}$. $\mathcal{A}$ picks the input vector σ, and the inputs for the honest parties are relayed to the challenger.

The challenger and $\mathcal{A}$ interact in a run of Π on input σ, with the challenger running Π on behalf of the honest parties, and $\mathcal{A}$ controlling the adversarial parties. At the end of the run, $\mathcal{A}$ outputs a bit b.

The ideal setting. This setting is defined with respect to two algorithms:

- An onion generation algorithm $\mathsf{GenOnions}$ takes as input the security parameter 1^λ, the public keys $\{\mathsf{pk}_{P_j}\}_{j=1}^N$ of all the parties, the bulletin B, the churn schedule C, the run number R, the identity P_i of an honest party, and the input σ_i for P_i; and outputs a set $\mathcal{O}_i^{(1)}$ of onions for P_i, i.e., $\mathcal{O}_i^{(1)} \leftarrow \mathsf{GenOnions}(1^\lambda, \{\mathsf{pk}_{P_j}\}_{j=1}^N, B, C, R, P_i, \sigma_i)$.
- A scheduling algorithm $\mathsf{ScheduleProcOnions}$ takes as input the security parameter 1^λ, the round number r, the identity P_i of an honest party, and the state $\mathsf{OnionBuffer}_i^{(r)}$ of P_i at round r; and outputs a set $\mathcal{O}_i^{(r)}$ of onions to be processed starting at round r and an updated state $\mathsf{OnionBuffer}_i^{(r+1)}$, i.e., $(\mathcal{O}_i^{(r)}, \mathsf{OnionBuffer}_i^{(r+1)}) \leftarrow \mathsf{ScheduleProcOnions}(1^\lambda, r, P_i, \mathsf{OnionBuffer}_i^{(r)})$.

$\mathsf{IdealGame}(\mathcal{A}, \mathsf{GenOnions}, \mathsf{ScheduleProcOnions}, \lambda)$ is parametrized by the adversary $\mathcal{A}$, the onion generation algorithm $\mathsf{GenOnions}$, the scheduling algorithm $\mathsf{ScheduleProcOnions}$, and the security parameter λ.

The game proceeds as follows. Like in the real setting, the adversary first picks Bad, B, C, R, and the keys $\{\mathsf{pk}_{P_j}\}_{j\in\mathsf{Bad}}$ for the adversarial parties, while the challenger runs KeyGen to generate the keys $\{\mathsf{pk}_{P_j}\}_{j\in[N]\setminus\mathsf{Bad}}$ for honest parties; and $\mathcal{A}$ determines the input vector $\sigma = (\sigma_1, \dots, \sigma_N)$ for the run.

The challenger and $\mathcal{A}$ interact in a run of Π on input σ, with the challenger acting as the honest parties, and $\mathcal{A}$ controlling the rest. In contrast to the real setting, the challenger doesn't run the protocol Π.

Instead, in the first round, for each honest party P_i, the challenger runs $\mathsf{GenOnions}(1^\lambda, \{\mathsf{pk}_{P_j}\}_{j=1}^N, B, C, R, P_i, \sigma_i)$ and sets P_i's initial state $\mathsf{OnionBuffer}_i^{(1)}$ to the output $\mathcal{O}_i^{(1)} \leftarrow \mathsf{GenOnions}(1^\lambda, \{\mathsf{pk}_{P_j}\}_{j=1}^N, B, C, R, P_i, \sigma_i)$. Then, still within the first round, for each honest party P_i, the challenger runs $\mathsf{ScheduleProcOnions}(1^\lambda, 1, P_i, \mathsf{OnionBuffer}_i^{(1)})$ to obtain a set $\mathcal{O}_i^{(1)} \subseteq \mathsf{OnionBuffer}_i^{(1)}$ of onions to be processed and an updated state $\mathsf{OnionBuffer}_i^{(2)}$. The challenger updates P_i's state to $\mathsf{OnionBuffer}_i^{(2)}$. For each onion $O \in \mathcal{O}_i^{(1)}$, the challenger initiates $\mathsf{ProcOnion}$ with P_i as the processing party and O as the onion to be processed and sends out the peeled onion $O_{1,i\to j}$ to its next destination $P_{1,i\to j}$ (whenever $\mathsf{ProcOnion}$ terminates).

In each subsequent round r, and for each honest party P_i, the challenger first adds the onions that P_i received in the previous round to $\mathsf{OnionBuffer}_i^{(r)}$. Then, the challenger runs $\mathsf{ScheduleProcOnions}(1^\lambda, r, P_i, \mathsf{OnionBuffer}_i^{(r)})$ to obtain $\mathcal{O}_i^{(r)}$ and $\mathsf{OnionBuffer}_i^{(r+1)}$. The challenger updates P_i's state to $\mathsf{OnionBuffer}_i^{(r+1)}$. For each $O \in \mathcal{O}_i^{(r)}$, the challenger initiates $\mathsf{ProcOnion}$ with P_i as the processing party and O as the onion to be processed and sends out the peeled onion $O_{r,i\to j}$ to its next destination $P_{r,i\to j}$ (whenever $\mathsf{ProcOnion}$ terminates).

At the end of the run, $\mathcal{A}$ outputs a bit b.

We define simulatablity using $\mathsf{RealGame}$ and $\mathsf{IdealGame}$ as follows:

Definition 4 (Simulatablity)**.** *An onion routing protocol Π is simulatable if for every p.p.t. adversary $\mathcal{A}$ there exist p.p.t. algorithms* ($\mathsf{GenOnions}, \mathsf{ScheduleProcOnions}$) *such that $\mathcal{A}$ can distinguish between* $\mathsf{RealGame}$ *and* $\mathsf{IdealGame}$ *with only negligible advantage, i.e.,*

$$\begin{aligned}&|\Pr[1 \leftarrow \mathsf{RealGame}(\mathcal{A}, \Pi, \lambda)]\\ &- \Pr[1 \leftarrow \mathsf{IdealGame}(\mathcal{A}, \Pi, \mathsf{GenOnions}, \mathsf{ScheduleProcOnions}, \lambda)]| = \mathsf{negl}(\lambda).\end{aligned}$$

5.3 From single-run to multi-run anonymity

Here, we prove that for simulatable onion routing protocols, single-run strong anonymity is equivalent to multi-run strong anonymity.

Theorem 2. *Let Π be a simulatable onion routing protocol with security parameter λ. For any $L = \mathsf{poly}(\lambda)$, Π is L-strongly anonymous from the active (resp. passive) adversary $\mathbb{A}$ with churn limit $c(N)$ if and only if it is single-run strongly anonymous from $\mathbb{A}$.*

Proof. It is evident that multi-run anonymity implies single-run anonymity since the former holds for any (polynomially bounded) number of runs, including one. Thus, to prove the theorem, it suffices to show that single-run anonymity implies multi-run anonymity. We do this using a hybrid argument.

Let Π be an onion routing protocol with security parameter λ that is single-run strongly anonymous against the active (resp. passive) adversary. Let $\mathcal{A}$ be any p.p.t. adversary from the class of active (resp. passive) adversaries.

Let $\mathsf{Experiment}_0$ be the anonymity game $\mathsf{StrongAnonGame}(\mathcal{A}, \Pi, L, \lambda)$ conditioned on the challenge bit b equaling zero, i.e., $b = 0$. Let $\sigma_0 = (\sigma_0^1, \ldots, \sigma_0^L)$ denote the sequence of input vectors that $\mathcal{A}$ chooses for the L runs in $\mathsf{Experiment}_0$; that is, σ_0^i is the input vector for the i^{th} run.

Likewise, let $\mathsf{Experiment}_1$ be $\mathsf{StrongAnonGame}(\mathcal{A}, \Pi, L, \lambda)$ when $b = 1$. Let $\sigma_1 = (\sigma_1^1, \ldots, \sigma_1^L)$ be the L input vectors in $\mathsf{Experiment}_1$.

We define a sequence of hybrids as follows. For all $1 \leq i \leq L+1$, let Hybrid_i be the experiment where the input vector for run j is σ_0^j if $j < i$, and otherwise, it is σ_1^j. Clearly, $\mathsf{Experiment}_0$ is the same as Hybrid_{L+1}, and $\mathsf{Experiment}_1$ is the same as Hybrid_1.

To complete the hybrid argument that Π is multi-run anonymous, we show that any two consecutive hybrids are distinguishable. To do so, we define another anonymity game, $\mathsf{FlipAnonGame}(\mathcal{A}, \Pi, \lambda, L, i)$, that we use only in this proof. This game is essentially the same as $\mathsf{StrongAnonGame}$ with the same parameters, except the challenger runs Π on σ_0 up to (but not necessarily including) run i and runs Π on σ_1 for the remaining runs when $b = 1$. The index i specifies where this switch from σ_0 to σ_1 happens. The challenger chooses $b \in \{0, 1\}$ uniformly at random. If $b = 0$, the first run with input σ_0 is run i. If $b = 1$, the first run with input σ_0 is run $i+1$. The adversary $\mathcal{A}$ makes a guess b' of whether the challenger switched in run i or in run $i+1$ and wins if $b' = b$.

To prove that consecutive hybrids are indistinguishable, we prove that $\mathcal{A}$ wins $\mathsf{FlipAnonGame}(\mathcal{A}, \Pi, \lambda, L, i)$ with only negligible advantage. Suppose there exists an index i such that $\mathcal{A}$ wins $\mathsf{FlipAnonGame}(\mathcal{A}, \Pi, \lambda, L, i)$ with non-negligible advantage. Then, we can construct a reduction $\mathcal{B}$ that uses $\mathcal{A}$ to "break" single-run strong anonymity. $\mathcal{B}$ goes between $\mathcal{A}$ and the challenger $\mathcal{C}$ in $\mathsf{StrongAnonGame}(\mathcal{B}, \Pi, \lambda, 1)$. We describe the interactions between $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ in terms of the phases in $\mathsf{StrongAnonGame}$.

The setup phase. During setup, the reduction $\mathcal{B}$ serves as a channel between the adversary $\mathcal{A}$ (of $\mathsf{FlipAnonGame}$) and the challenger (of the single-run anonymity game). $\mathcal{A}$ sends the set of adversarial parties to the reduction $\mathcal{B}$; $\mathcal{B}$ relays this to $\mathcal{C}$. $\mathcal{C}$ sends the honest parties' public keys to $\mathcal{B}$; $\mathcal{B}$ relays them to $\mathcal{A}$. $\mathcal{A}$ sends the adversarial parties' public keys to $\mathcal{B}$; $\mathcal{B}$ relays them to $\mathcal{C}$. During the first query phase, $\mathcal{A}$ can send $\mathsf{ProcOnion}$ queries to $\mathcal{B}$. Whenever $\mathcal{A}$ sends a $\mathsf{ProcOnion}$ query with a bulletin B and a churn schedule C (such that the number of offline parties is at most $c(N)$), an onion O, a processing party P, and a round number r, $\mathcal{B}$ relays the query to $\mathcal{C}$ and replies to $\mathcal{A}$ with $\mathcal{C}$'s response.

The challenge phase. Since (from the hypothesis) Π is simulatable, it follows that there exist efficient algorithms $(\mathsf{GenOnions}, \mathsf{ScheduleProcOnions})$ such that no efficient algorithm can tell whether

$\mathcal{B}$ is running the protocol Π, or simulating the run by running GenOnions and ScheduleProcOnions and submitting ProcOnion queries to the challenger, instead.

For each run j of the challenge phase, $\mathcal{A}$ sends the run parameters $(B_j, C_j, \sigma_0^j, \sigma_1^j)$ to $\mathcal{B}$. If $j < i$, $\mathcal{B}$ simulates a run of Π with parameters B_j, C_j, and σ_0. If $j > i$, $\mathcal{B}$ simulates the run with parameters B_j, C_j, and σ_1 instead. The i^{th} run is the challenge run in FlipAnonGame. In this run, $\mathcal{B}$ uses these parameters in its challenge run, relaying them all to $\mathcal{C}$ and serving as a channel between $\mathcal{A}$ and $\mathcal{C}$ in running Π on either $\sigma_0^{j=i}$ or $\sigma_1^{j=i}$, depending on the challenge bit b chosen by $\mathcal{C}$.

The final phase. During the second query phase, $\mathcal{A}$ is again allowed to submit ProcOnion queries. Whenever $\mathcal{A}$ sends a ProcOnion query with a bulletin B and a churn schedule C (such that the number of offline parties is at most $c(N)$), an onion O (where O was not produced by an honest party during the challenge phase), a processing party P, and a round number r, $\mathcal{B}$ relays the query to $\mathcal{C}$ and replies to $\mathcal{A}$ with $\mathcal{C}$'s response. Finally, $\mathcal{A}$ makes its guess b' for FlipAnonGame and passes b' to $\mathcal{B}$. If $\mathcal{A}$ guesses $b' = 0$, this means that $\mathcal{A}$ suspects that the first run with input σ_1 is run i. Thus if $\mathcal{A}$'s guess is correct, the input in the challenge run i for StrongAnonGame was likely σ_1, and $\mathcal{B}$ should output 1. Thus, $\mathcal{B}$ outputs the opposite of b' (i.e., 1 if $b' = 0$ and 0 if $b' = 1$).

Since $\mathcal{B}$ essentially wins whenever $\mathcal{A}$ wins, we conclude that no efficient adversary can win FlipAnonGame with non-negligible advantage.

□

An immediate consequence of Theorem 2 is that:

Corollary 1. *If* Π *is a simulatable onion routing protocol, then, in the static setting (i.e. for* $c(N) = 0$*),* Π *is multi-run strongly anonymous from the active (resp. passive) adversary iff it is single-run strongly anonymous from the active (resp. passive) adversary.*

6 Multi-run strongly anonymous onion routing

Note that we can turn any weakly anonymous onion routing protocol strongly anonymous by using a sufficiently secure onion encryption scheme (e.g., any scheme that realizes Camenisch and Lysyanskaya's onion ideal functionality [9]). Thus from Corollary 1, in the static setting, any simulatable onion routing protocol shown to be single-run anonymous is also anonymous over multiple runs. For example, Ando, et al. [3] proved that their protocol Π_p is anonymous from the passive adversary in the static setting; since Π_p is simulatable (Lemma 2), this means that running Π_p multiple times is still anonymous.

However, Π_p is not guaranteed to work in the dynamic setting; e.g., when the churn limit is linear in the number of participants, Π_p either fails to deliver any messages or is not anonymous (Theorem 3). In this section, we show that we can make Π_p multi-run anonymous from the passive adversary with this churn limit if we use poly onion encryption instead of regular onion encryption (Theorem 4).

The protocol Π_p. Ando, Lysyanskaya, and Upfal [3] showed that the simple protocol Π_p is weakly anonymous from the passive adversary in the (simple I/O) static setting. For this protocol, there are N users that send and receive messages: $\mathcal{P} = P_1, \ldots, P_N$; and $n < N$ mix-servers that serve as intermediaries on routing paths: $\mathcal{S} = S_1, \ldots, S_n$. During the onion-forming phase of a protocol execution, each user P_i forms an onion to carry the message $m_{i \to j}$ to his recipient P_j. Specifically, P_i first picks a random sample $T_1, \ldots, T_\ell$ from the set $\mathcal{S}$ of mix-servers (with replacement), i.e., $T_1, \ldots, T_\ell \leftarrow\$ \mathcal{S}$, then generates an onion using the message $m_{i \to j}$, the path $(T_1, \ldots, T_\ell, P_j)$, and the public keys for all the parties on the path. During the first round of the execution phase, the users send the generated onions to their first locations (i.e., first parties on the paths). During all subsequent rounds, each party peels the onions from the previous round and sends the peeled onions to their next locations or outputs the received messages for the final round.

Ando et al. [3] proved that Π_p is anonymous from the passive adversary that corrupts up to a constant $0 \leq \beta_1 < 1$ fraction of the servers when both the server load (the average number of onions per server per round) $\frac{N}{n}$ and the number ℓ of rounds are at least polylog in the security parameter λ. However, this result holds in the static setting without any churn.

For all the results below, let $\mathsf{polylog}(\lambda)$ denote any polylog function in λ. Additionally, when we say that a server is *online*, we mean that it is online throughout the protocol run; otherwise, the server is *offline*.

Theorem 3. *When the churn limit is $c(N) = \beta_2 N$ where $0 < \beta_2 \leq 1$ is any positive constant, a single run of Π_p either fails to deliver any message with overwhelming probability, or else it is not (single-run weakly) anonymous.*

Proof. Let λ denote the security parameter.

Case 1: when the length of the routing path $\ell \geq \mathsf{polylog}(\lambda)$. Let P_i be any sender. Let E_i be the event that the onion generated by P_i makes it to the recipient of P_i. This is the event that all of the intermediaries $T_1, \ldots, T_\ell$ that P_i picks are online. Since each T_j is online with probability $(1 - \beta_2)$, $\Pr[E_i] = (1 - \beta_2)^\ell \leq (1 - \beta_2)^{\mathsf{polylog}(\lambda)}$. In other words, E_i occurs with negligible probability. By a union bound, the probability that *any* of the $\ell = \mathsf{poly}(\lambda)$ messages gets through is also negligibly small. Thus, in this case, Π_p fails to route any message.

Case 2: when the length of the routing path $\ell < \mathsf{polylog}(\lambda)$. We know from previous work [12, 17, 18] that with a passive adversary corrupting a constant fraction of the parties, no onion routing protocol with fewer than polylog rounds is anonymous. □

We just showed that the protocol Π_p, using standard onion encryption, doesn't work when the churn limit is linear in the number of participants. However, we can guarantee anonymous message delivery even with this churn limit by modifying Π_p so that it uses Poly Onion Encryption instead.

Π_p with Poly Onion Encryption. To generate a poly onion, each sender P_i first randomly chooses κ candidates $\mathcal{P}_h = (P_{h,1}, \ldots, P_{h,\kappa})$ for each intermediary hop h of the path and $\kappa - 1$ candidates $(P_{\ell+1,2}, \ldots, P_{\ell+1,\kappa})$ for the final $(\ell + 1)^{st}$ hop. P_i then randomly chooses ν helpers $\mathcal{Q}_h = (Q_{h,1}, \ldots, Q_{h,\nu})$ for each hop h of the path, i.e., $P_{1,1}, \ldots, P_{1,\kappa}, \ldots, P_{\ell+1,2}, \ldots, P_{\ell+1,\kappa}, Q_{1,1}, \ldots, Q_{1,\nu}, \ldots, Q_{\ell,1}, \ldots, Q_{\ell,\nu} \leftarrow\$ \mathcal{S}$. P_i then forms an onion using the message m to her recipient $P_{\ell+1,1}$, the candidates $(\mathcal{P}_1, \ldots, \mathcal{P}_\ell, (P_{\ell+1,1}, P_{\ell+1,2}, \ldots, P_{\ell+1,\kappa}))$, the helpers $(\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell)$, and all the required public keys.

For the analysis below, we will make the simplifying assumption that $\mathsf{ProcOnion}$ runs within a single round since making this assumption doesn't change the results. We use the committee threshold parameter $\alpha = \frac{1}{2}$. By the security of Poly Onion Encryption (Theorem 1), onions formed by honest parties "mix" in hop h when the first online candidate in $\mathcal{P}_h$ is honest (event E_3 in the proof), and fewer than $\frac{1}{2}$ of the members of $\mathcal{Q}_{h-1}$ are corrupted (event E_4 in the proof). Note that these conditions are stronger than what is required for security to hold.

6.1 Poly Π_p is multi-run anonymous in the presence of churn

For all the results below, let *Poly* Π_p be the protocol Π_p modified to use Poly Onion Encryption instead of regular onion encryption with the following parameter settings: security parameter λ, length of the routing path $\ell \geq \mathsf{polylog}(\lambda)$, and number of candidates per hop $\kappa \geq \mathsf{polylog}(\lambda)$, and number of helpers per hop $\nu \geq \mathsf{polylog}(\lambda)$.

Towards showing that Poly Π_p is multi-run anonymous when the churn limit is linear in the number of mix-servers, we now prove that Poly Π_p is both single-run anonymous in the setting with churn (Definition 3) and simulatable (Definition 4).

Lemma 1. *Poly Π_p is single-run (strongly) anonymous from the passive adversary who corrupts up to a constant $0 \leq \beta_1 < 1$ fraction of the mix-servers, when the churn limit is $c(N) = \beta_2 N$ and $0 \leq \beta_1 + \beta_2 < \frac{1}{2}$ is a constant. Moreover, it delivers all messages with overwhelming probability.*

Proof. An onion is dropped at an intermediary $P_{h,j} \in \mathcal{P}_h$ due to churn only if all of the candidates $\mathcal{P}_h$ are offline (event E_1), or at least $\frac{\nu}{2}$ of the helpers $\mathcal{Q}_{h-1}$ are offline (event E_2). The probability of E_1 is negligibly small since the probability that each randomly chosen candidate is offline is bounded above by $\frac{1}{2}$. We can show that the probability of E_2 is also negligibly small by using a Chernoff bound for Poisson trials [25, Corollary 4.6]; with overwhelming probability, the fraction of offline parties in the committee is arbitrarily close to the expected value, which is strictly less than $\frac{\nu}{2}$. Since E_1 and E_2 occur with only negligible probabilities, this onion (layer) at $P_{i,j}$ is not dropped. Since the total number of onion layers is polynomially bounded in the security parameter, by a union bound, it follows that with overwhelming probability, no onion is dropped.

Since no onions are dropped, we can apply the proof of weak anonymity of Π_p from Ando et al. [3], with a slight modification. In that proof, mixing occurs at an intermediary server as long as that server is honest. This happens with constant probability in Ando et al.'s construction. With Poly Onion Encryption, mixing occurs when the first online candidate in $\mathcal{P}_h$ is honest (event E_3), and fewer than $\frac{1}{2}$ of the members of $\mathcal{Q}_{h-1}$ are corrupted (event E_4). The probability that any random party is both honest and online is at least $1 - \beta_1 - \beta_2 > \frac{1}{2}$ since, in the most pessimistic scenario, the adversary chooses the set of corrupted servers to be disjoint from the set of offline servers. Thus, E_3 happens with probability at least $\frac{1}{2}$. Similar to the analysis of $\bar{E_2}$, from a Chernoff bound, E_4 also occurs with overwhelming probability. Thus, the proof of weak anonymity of Π_p still holds for Poly Π_p, and all onions will be untraceable to their senders by the time they reach their last intermediaries. An onion may be dropped in its final relay to its recipient with non-negligible probability; however, it is already untraceable to its sender at this point. This proves that Poly Π_p is single-run weakly anonymous. The protocol is also single-run *strongly* anonymous since it is constructed with a sufficiently strong encryption scheme that is poly-onion secure. □

Lemma 2. *Poly Π_p is simulatable.*

Proof. We describe algorithms $\mathsf{GenOnions}$ and $\mathsf{ScheduleProcOnions}$ for which Poly Π_p is simulatable.

Defining $\mathsf{GenOnions}$. Recall that $\mathsf{GenOnions}$ takes as input the security parameter 1^λ, the public keys $\{\mathsf{pk}_{P_k}\}_{k=1}^N$ of all the parties, the bulletin B, the churn schedule C, the run number R, the identity P_i of an honest party, and the input σ_i for P_i; and outputs a set $\mathcal{O}_i^{(1)}$ of onions for P_i, i.e., $\mathcal{O}_i^{(1)} \leftarrow \mathsf{GenOnions}(1^\lambda, \{\mathsf{pk}_{P_k}\}_{k=1}^N, B, C, R, P_i, \sigma_i)$. Let P_j denote the recipient and let m denote the message for that recipient included in σ_i. $\mathsf{GenOnions}$ first generates a list of candidate lists $\mathcal{P}_1, \ldots, \mathcal{P}_\ell, \mathcal{P}_{\ell+1}$, where $P_{\ell,1} = P_j$, and all other candidates is chosen independently and uniformly at random. $\mathsf{GenOnions}$ then generates a list of committees $\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell$, where each party in each list is chosen independently and uniformly at random. Let $\{\mathsf{pk}_{P_k}\}_{k \in \mathcal{P} \cup \mathcal{Q}}$ denote the set of public keys of all parties in some candidate list $\mathcal{P}_j$ or some committee $\mathcal{Q}_j$. Each candidate list has length κ, and each committee has size ν, where κ and ν are our chosen Poly Onion Encryption parameters. $\mathsf{GenOnions}$ then runs $\mathsf{FormOnion}$ to obtain $((\mathcal{O}_{1,1}, \ldots, \mathcal{O}_{1,\kappa}), \ldots, (\mathcal{O}_{\ell,1}, \ldots, \mathcal{O}_{\ell,\kappa}) \leftarrow \mathsf{FormOnion}(m, R, (\mathcal{P}_1, \ldots, \mathcal{P}_{\ell+1}), (\mathcal{Q}_1, \ldots, \mathcal{Q}_{\ell+1}), \{\mathsf{pk}_{P_k}\}_{k \in \mathcal{P} \cup \mathcal{Q}})$.

The output $\mathcal{O}_i^{(1)}$ of GenOnions should be the singleton containing an onion O_0 such that processing it right away has the same effect as the sender P_i sending the first onion $O_{1,u}$ to the first available candidate $P_{1,u} \in \mathcal{P}_1$ for the first hop. We can construct O_0 from the onion $O_{1,1}$ for the preferred candidate $P_{1,1}$ by "wrapping" it with an extra layer of encryption using as parameters, the candidate lists $\mathcal{P}_0 = (P_i, \ldots, P_i)$ and $\mathcal{P}_1$ and the helper list $\mathcal{Q}_0 = \mathcal{P}_0 = (P_i, \ldots, P_i)$. (See "Wrapping an onion" in Section 4.2 for the exact details on how to do this.)

Defining ScheduleProcOnions. Recall that ScheduleProcOnions takes as input the security parameter 1^λ, the round number r, the identity P_i of an honest party, and the state $\mathsf{OnionBuffer}_i^{(r)}$ of P_i at round r; and outputs a set of onions $\mathcal{O}_i^{(r)}$ to be processed and sent out during round r and an updated state $\mathsf{OnionBuffer}_i^{(r+1)}$. We define ScheduleProcOnions for Π_p to return all onions on $\mathsf{OnionBuffer}_i^{(r)}$ to be processed immediately, and to return an empty buffer $\mathsf{OnionBuffer}_i^{(r+1)}$ for the next round.

Simulatability. Π_p is simulatable using GenOnions and ScheduleProcOnions as defined here because they are defined identically to the honest parties' behavior in the actual protocol. In Π_p, each party sends its onion on the first round, processes onions immediately when it receives them, and forwards onions immediately when processed. Thus, RealGame is identical to IdealGame. □

We just showed that Poly Π_p is single-run (strongly) anonymous (Lemma 1) and simulatable (Lemma 2). Thus, from Theorem 2, it follows that:

Theorem 4. *Poly Π_p is multi-run (strongly) anonymous from the passive adversary who corrupts up to a constant $0 \le \beta_1 < 1$ fraction of the mix-servers, when the churn limit is $c(N) = \beta_2 N$ and $0 \le \beta_1 + \beta_2 < \frac{1}{2}$ is a constant. Moreover, it delivers all messages with overwhelming probability.*

Acknowledgments

We thank Eli Upfal for helpful discussions. This research was supported in part by NSF grant CCF-2107187, by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under award number DE-SC-0001234, by a grant from the Columbia-IBM center for Blockchain and Data Transparency, by JPMorgan Chase & Co., and by LexisNexis Risk Solutions. Any views or opinions expressed herein are solely those of the authors listed.

References

[1] Tor directory protocol, version 3. `https://gittweb.torproject.org/torspec.git/plain/dir-spec.txt`.

[2] Megumi Ando and Anna Lysyanskaya. Cryptographic shallots: A formal treatment of repliable onion encryption. In *Theory of Cryptography Conference*, pages 188–221. Springer, 2021.

[3] 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.

[4] Megumi Ando, Anna Lysyanskaya, and Eli Upfal. On the complexity of anonymous communication through public networks. In *2nd Conference on Information-Theoretic Cryptography (ITC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[5] John Augustine, Gopal Pandurangan, Peter Robinson, Scott T. Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In Venkatesan Guruswami, editor, *56th FOCS*, pages 350–369. IEEE Computer Society Press, October 2015.

[6] 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.

[7] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. Anoa: A framework for analyzing anonymous communication protocols. In *2013 IEEE 26th Computer Security Foundations Symposium*, pages 163–178. IEEE, 2013.

[8] Matt Blaze, John Ioannidis, Angelos D Keromytis, Tal Malkin, and Aviel D Rubin. Anonymity in wireless broadcast networks. *IJ Network Security*, 8(1):37–51, 2009.

[9] 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.

[10] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[11] Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrain Perrig. Hornet: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1441–1454, 2015.

[12] Miranda Christ. New lower bounds on the complexity of provably anonymous onion routing. *Undergraduate honors thesis, Brown University, Providence, RI*, 2020.

[13] 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.

[14] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002.

[15] 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.

[16] 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.

[17] 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 (SP)*, pages 108–126. IEEE, 2018.

[18] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Comprehensive anonymity trilemma: User coordination is not enough. *Proceedings on Privacy Enhancing Technologies*, 2020(3):356–383, 2020.

[19] 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.

[20] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Profiling a million user dht. IMC '07, New York, NY, USA, 2007. Association for Computing Machinery.

[21] Jan Iwanik, Marek Klonowski, and Miroslaw Kutyłowski. Duo-onions and hydra-onions—failure and adversary resistant onion protocols. In *Communications and Multimedia Security*, pages 1–15. Springer, 2005.

[22] Christiane Kuhn, Martin Beck, and Thorsten Strufe. Breaking and (partially) fixing provably secure onion routing. In *2020 IEEE Symposium on Security and Privacy*, pages 168–185. IEEE Computer Society Press, May 2020.

[23] Christiane Kuhn, Dennis Hofheinz, Andy Rupp, and Thorsten Strufe. Onion routing with replies. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 573–604. Springer, 2021.

[24] 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*, SOSP '17, page 406–422, New York, NY, USA, 2017. Association for Computing Machinery.

[25] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[26] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy*, pages 183–195. IEEE Computer Society Press, May 2005.

[27] Charles Rackoff and Daniel R. Simon. Cryptographic defense against traffic analysis. In *25th ACM STOC*, pages 672–681. ACM Press, May 1993.

[28] Charles Rackoff and Daniel R Simon. Cryptographic defense against traffic analysis. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 672–681, 1993.

[29] Lucas Ropek. Someone is running hundreds of malicious servers on the Tor network and might be de-anonymizing users. `https://tinyurl.com/2p999e8e`, December 2021.

[30] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. IMC '06, New York, NY, USA, 2006. Association for Computing Machinery.

[31] 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.

[32] 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*, SOSP '17, page 423–440, New York, NY, USA, 2017. Association for Computing Machinery.

[33] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. *Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis*, page 137–152. Association for Computing Machinery, New York, NY, USA, 2015.

A Security of Poly Onion Encryption

A.1 $\mathsf{Hybrid}^1 \approx \mathsf{Hybrid}^2$

Fix an encryption $c = \mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\dots0)$ unknown to the adversary. By security of Enc, the probability that the adversary queries c to the oracle is negligible.

A.2 $\mathsf{Hybrid}^2 \approx \mathsf{Hybrid}^3$

Recall that in Hybrid^2, we form the challenge onion by first forming $\mathcal{O}_{i+2}$, then wrapping O^+_{i+2} to get $\mathcal{O}_1$. Hybrid^3 is identical to Hybrid^2, but we change the share $\sigma_{i,j}$ of each member $Q_{i,j}$ of committee $\mathcal{Q}_i$ to a share of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\dots0)$ instead of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(k_{i+1}, \text{role}(P^-_{i+1}), P^+_{i+2})$ and construct the corresponding authentication tag $T_{i,j}$ for this modified share.

More precisely, in Hybrid^3, the challenger forms O^+_{i+2} as in Hybrid^2, then wraps it to obtain $\mathcal{O}_{i+1}$. The challenger then wraps O^+_{i+1}, but replaces the shares σ_i in the resulting block H^1_{i+1} of O^+_i with shares of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\dots0)$. It forms the set of tags T_{i+1} using these modified shares. Let this modified wrapping be $\mathcal{O}_i$. The challenger then wraps O^+_i to obtain $\mathcal{O}_i$. Let $(O^-_{i+1})'$ denote O^-_{i+1}, but with K^1_{i+1} replaced by $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\dots0)$. We modify the $\mathsf{ProcOnion}$ such that in the second query phase, if asked to process $(O^-_{i+1})'$ by P^-_{i+1}, it instead processes O^-_{i+1} by P^-_{i+1}.

We split the proof that Hybrid^2 and Hybrid^3 are indistinguishable into two cases. In the first case, P^+_{i+1} is honest and online. The proof for this case involves three subproofs. First, we show that in H^1_i, the shares of the honest members of $\mathcal{Q}_i$ can be replaced by random strings. Second, we show that in H^1_i, the shares of the corrupted members of $\mathcal{Q}_i$ can be replaced by shares of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\dots0)$. Third, we show that in H^1_i, the honest members' random strings can be replaced by shares of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\dots0)$.

In the second case, P^+_{i+1} is honest and offline; since the security predicate must be satisfied, this means that P^-_{i+1} is honest. Here, we can prove that $\mathsf{Hybrid}^2 \approx \mathsf{Hybrid}^3$ by a reduction to the security of public key encryption for party P^-_{i+1}.

Proposition 1. *If P^+_{i+1} is honest and online,* $\mathsf{Hybrid}^2 \approx \mathsf{Hybrid}^3$.

Proof. First, we show that we can replace the share of each honest committee member $Q_{i,j}$ with a random string of the same length by a reduction to the security of public key encryption under $\mathsf{pk}_{Q_{i,j}}$. Thus, by a hybrid argument, we can replace all committee members' shares.

Suppose that there exists $\mathcal{A}$ that can distinguish between Hybrid^2 with a subset $S \subseteq \mathcal{Q}_i$ of the honest committee members' shares replaced with random strings, and Hybrid^2 with $S \cup Q_{i,j} \subseteq \mathcal{Q}_i$ of the honest committee members' shares replaced with random strings. We will construct $\mathcal{B}$ that can break the security of Enc.

1. $\mathcal{B}$ generates the public and private keys for all parties except for $Q_{i,j}$, whose public key $\mathcal{B}$ obtains from the challenger. $\mathcal{B}$ sends the public keys for all parties to $\mathcal{A}$.
2. $\mathcal{A}$ sends the set of corrupted parties Bad, the bulletin B, the churn schedule C, and the public keys for Bad to $\mathcal{B}$.

3. $\mathcal{B}$ gives $\mathcal{A}$ oracle access to ProcOnion on behalf of the honest parties.
4. $\mathcal{B}$ receives from $\mathcal{A}$ the challenge parameters: a path length ℓ, a message m, a routing position $i < \ell$, a round r, committees $\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell$, and candidates $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$.
5. $\mathcal{B}$ samples keys $k_1, \ldots, k_{i+1}$. $\mathcal{B}$ forms O_{i+2}^+ and wraps it once using k_{i+1} to form $\mathcal{O}_{i+1}$ like in Hybrid^2. $\mathcal{B}$ creates shares of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \text{role}(P_{i+1}^-), P_{i+2}^+)$ for all parties in $\mathcal{Q}_i$. Let $\sigma_{i,j}$ denote the share of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \text{role}(P_{i+1}^-), P_{i+2}^+)$ for party $Q_{i,j}$. $\mathcal{B}$ also samples uniformly random strings of the same length of the shares for each party in $\mathcal{Q}_i$. Let $r_{i,j}$ denote the random string of length $|\sigma_{i,j}|$ for party $Q_{i,j}$.
6. $\mathcal{B}$ sends messages $m^0 = (P_{i+1}^+, P_{i+1}^-, \sigma_{i,j})$ and $m^1 = (P_{i+1}^+, P_{i+1}^-, r_{i,j})$ to the challenger and receives back ciphertext $c = \mathsf{Enc}_{\mathsf{pk}_{Q_{i,j}}}(m^b)$. $\mathcal{B}$ samples a tag $T_{i,j} \leftarrow \mathsf{Tag}_{k_i}(\sigma_{i,j})$. $\mathcal{B}$ continues to form $\mathcal{O}_i$, using the random strings (and their corresponding authentication tags) instead of the shares for the inputs for parties in S, using c and $T_{i,j}$ in the input for $Q_{i,j}$, and using the shares of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \text{role}(P_{i+1}^-), P_{i+2}^+)$ for the corrupted committee members. $\mathcal{B}$ then wraps O_i^+ to obtain $\mathcal{O}_1$, as in Hybrid^2.
7. $\mathcal{B}$ gives $\mathcal{O}_1$ to $\mathcal{A}$ and gives $\mathcal{A}$ query access to ProcOnion on behalf of the honest parties. This query access is modified such that if $\mathcal{A}$ queries $(O_{i+1}^-)'$ to be processed by P_{i+1}^-, $\mathcal{B}$ instead runs ProcOnion by P_{i+1}^- on O_{i+1}^-.
   $\mathcal{B}$ does not need to decrypt c in order to simulate this query access. Because P_{i+1}^+ is honest and online in the challenge rounds, if $Q_{i,j}$ receives c as an input during the ProcOnion protocol, it simply returns $\perp$.
8. $\mathcal{B}$ submits the guess b' returned by $\mathcal{A}$.

Since $\mathcal{A}$ can distinguish between the two scenarios, $\mathcal{B}$ can determine which message was chosen by the challenger with non-negligible probability. By repeating this argument, we can replace all of the honest committee members' shares of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \text{role}(P_{i+1}^-), P_{i+2}^+)$ with random strings.

Next, we show that we can replace all corrupted parties' shares of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \text{role}(P_{i+1}^-), P_{i+2}^+)$ with shares of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(00\ldots0)$.

Recall that the secret sharing protocol we use has the security property that if the adversary has fewer than $\alpha \cdot \nu$ shares (α is the committee threshold, and ν is the committee size), it cannot distinguish between whether the shares are of a message m^0 or a message m^1. There are fewer than $\alpha \cdot \nu$ corrupted committee members in $\mathcal{Q}_i$, as ensured by the security predicate.

Suppose the honest parties' shares are replaced with random strings, and there exists an adversary $\mathcal{A}$ that can distinguish between whether the corrupted parties' shares are of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \text{role}(P_{i+1}^-), P_{i+2}^+)$ or $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(00\ldots0)$. We will construct $\mathcal{B}$ that can break the security of the secret sharing scheme.

1. $\mathcal{B}$ generates the public and private keys for all parties. $\mathcal{B}$ sends the public keys for all parties to $\mathcal{A}$.
2. $\mathcal{A}$ sends the set of corrupted parties Bad, the bulletin B, the churn schedule C, and the public keys for Bad to $\mathcal{B}$.
3. $\mathcal{B}$ gives $\mathcal{A}$ oracle access to ProcOnion on behalf of the honest parties.
4. $\mathcal{B}$ receives from $\mathcal{A}$ the challenge parameters: a path length ℓ, a message m, a routing position $i < \ell$, a round r, committees $\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell$, and candidates $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$.
5. $\mathcal{B}$ samples keys $k_1, \ldots, k_{i+1}$. $\mathcal{B}$ forms O_{i+2}^+ and wraps it once using k_{i+1} to form $\mathcal{O}_{i+1}$ like in Hybrid^2. $\mathcal{B}$ creates shares of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \text{role}(P_{i+1}^-), P_{i+2}^+)$ for all parties in $\mathcal{Q}_i$. Let $\sigma_{i,j}^0$ denote the share of $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^-}}(k_{i+1}, \text{role}(P_{i+1}^-), P_{i+2}^+)$ for party $Q_{i,j}$. $\mathcal{B}$ also samples uniformly

random strings of the same length of the shares for each party in $\mathcal{Q}_i$.

6. $\mathcal{B}$ sends $m^0 = \mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(k_{i+1}, \text{role}(P^-_{i+1}), P^+_{i+2})$ and $m^1 = \mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\ldots0)$ to the challenger and receives back shares σ^b_i of m^b for all parties in $\mathcal{Q}_i$. (Note that $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\ldots0)$ is the same encryption of $00\ldots0$ used to define $(O^-_{i+1})'$). $\mathcal{B}$ uses the corrupted parties' shares in σ^b_i to create their inputs, and it uses the honest parties' random strings to create their inputs. $\mathcal{B}$ forms $\mathcal{O}_i$ by wrapping O^+_{i+1}, with these modified inputs. $\mathcal{B}$ then wraps O^+_i as in Hybrid^2 to obtain $\mathcal{O}_1$.
7. $\mathcal{B}$ gives $\mathcal{O}_1$ to $\mathcal{A}$ and gives $\mathcal{A}$ query access to $\mathsf{ProcOnion}$ on behalf of the honest parties. This query access is modified such that if $\mathcal{A}$ queries $(O^-_{i+1})'$ to be processed by P^-_{i+1}, $\mathcal{B}$ instead runs $\mathsf{ProcOnion}$ by P^-_{i+1} on O^-_{i+1}.
8. $\mathcal{B}$ submits the guess b' returned by $\mathcal{A}$.

Thus, the adversary cannot distinguish between whether the corrupted parties' shares are replaced with shares of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\ldots0)$.

Finally, we can switch the honest parties' random strings to shares of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\ldots0)$ by the same reduction used to switch their shares to random strings. Thus, we arrive at Hybrid^3.

□

Proposition 2. *If P^+_{i+1} is honest and offline,* $\mathsf{Hybrid}^2 \approx \mathsf{Hybrid}^3$.

Proof. We reduce to CCA2-security of public key encryption under $\mathsf{pk}_{P^-_{i+1}}$. By definition of the security predicate, if P^+_{i+1} is honest and offline, P^-_{i+1} must be honest and thus the adversary does not know $\mathsf{sk}_{P^-_{i+1}}$.

Suppose that there exists $\mathcal{A}$ that can distinguish between Hybrid^2, where each $\sigma_{i,j}$ is a share of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(k_{i+1}, \text{role}(P^-_{i+1}), P^+_{i+2})$, and Hybrid^3, where each $\sigma_{i,j}$ is a share of $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\ldots0)$. We will construct $\mathcal{B}$ that can break the security of Enc.

1. $\mathcal{B}$ generates the public and private keys for all parties except for $Q_{i,j}$, whose public key $\mathcal{B}$ obtains from the challenger. $\mathcal{B}$ sends the public keys for all parties to $\mathcal{A}$.
2. $\mathcal{A}$ sends the set of corrupted parties Bad, the bulletin B, the churn schedule C, and the public keys for Bad to $\mathcal{B}$.
3. $\mathcal{B}$ gives $\mathcal{A}$ oracle access to $\mathsf{ProcOnion}$ on behalf of the honest parties.
4. $\mathcal{B}$ receives from $\mathcal{A}$ the challenge parameters: a path length ℓ, a message m, a routing position $i < \ell$, a round r, committees $(\mathcal{Q}_1, \ldots, \mathcal{Q}_{\ell+1})$, and candidates $(\mathcal{P}_1, \ldots, \mathcal{P}_{\ell+1})$ such that the security predicate holds.
5. $\mathcal{B}$ samples keys $k_1, \ldots, k_{i+1}$. $\mathcal{B}$ forms O^+_{i+2} and wraps it once using k_{i+1} to form $\mathcal{O}_{i+1}$ like in Hybrid^1.
6. $\mathcal{B}$ sends messages $m^0 = (k_{i+1}, \text{role}(P^-_{i+1}), P^+_{i+2})$ and $m^1 = 00\ldots0$ to the challenger and receives back ciphertext $c = \mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(m^b)$. $\mathcal{B}$ creates shares of c, with one share $\sigma_{i,j}$ for each committee member $Q_{i,j}$. $\mathcal{B}$ forms $\mathcal{O}_i$ using these shares. $\mathcal{B}$ then wraps O^+_i to obtain $\mathcal{O}_1$, as in Hybrid^1.
7. $\mathcal{B}$ sends $\mathcal{O}_1$ to $\mathcal{A}$ and gives $\mathcal{A}$ modified query access to $\mathsf{ProcOnion}$ on behalf of the honest parties. Let $(O^-_{i+1})'$ equal O^-_{i+1}, with K^1_{i+1} replaced by c. If $\mathcal{A}$ queries $(O^-_{i+1})'$ to be processed by P^-_{i+1}, $\mathcal{B}$ instead runs the $\mathsf{ProcOnion}$ protocol on O^-_{i+1}.
8. $\mathcal{B}$ submits the guess b' returned by $\mathcal{A}$.

Remark. In step 7, although $\mathcal{B}$ cannot query c to be decrypted by the challenger, it can still simulate query access to $\mathsf{ProcOnion}$. Since we use encryption with tags, the honest party P_{i+1}^{-} will only process an onion with $K_1 = c$ when the remaining blocks are from the challenge onion O_{i+1}^{-}. In this case, we defined the $\mathsf{ProcOnion}$ oracle to run the processing protocol on O_{i+1}^{-}, which does not require decrypting c since $\mathcal{B}$ formed $\mathcal{O}_{i+1}^{-}$ and thus knows its contents.

□

A.3 $\mathsf{Hybrid}^3 \approx \mathsf{Hybrid}^4$

Hybrid^4 is identical to Hybrid^3, except that k_{i+1} is changed to $00\dots0$ in the key block K_{i+1}^1. We prove that Hybrid^3 and Hybrid^4 are indistinguishable by a reduction to the security of Enc, the public key encryption scheme used in the key block.

Proof. Suppose that there exists $\mathcal{A}$ that can distinguish between Hybrid^3 and Hybrid^4. We construct $\mathcal{B}$ as follows.

1. $\mathcal{B}$ generates the public and private keys for all parties except for P_{i+1}, whose public key it obtains from the challenger. $\mathcal{B}$ sends the public keys for all parties to $\mathcal{A}$.
2. $\mathcal{A}$ sends the set of corrupted parties Bad, the bulletin B, the churn schedule C, and the public keys for Bad to $\mathcal{B}$.
3. $\mathcal{B}$ gives $\mathcal{A}$ oracle access to $\mathsf{ProcOnion}$ on behalf of the honest parties.
4. $\mathcal{B}$ receives from $\mathcal{A}$ the challenge parameters: a path length ℓ, a message m, a routing position $i < \ell$, a round r, committees $\mathcal{Q}_1, \dots, \mathcal{Q}_\ell$, and candidates $\mathcal{P}_1, \dots, \mathcal{P}_\ell$.
5. $\mathcal{B}$ constructs $\mathcal{O}_{i+2}$ using $\mathsf{FormOnion}$ given the challenge parameters, as in Hybrid^3. $\mathcal{B}$ samples keys $k_1, \dots, k_{i+1}$ for constructing the rest of the onion.
6. $\mathcal{B}$ chooses messages $m^0 = (k_{i+1}, role(P_{i+1}), P_{i+2})$ and $m^1 = (00\dots0, role(P_{i+1}), P_{i+2})$ and sends them to the challenger. $\mathcal{B}$ obtains $c = \mathsf{Enc}_{\mathsf{pk}_{P_{i+1}}}(m^b)$ from the challenger. $\mathcal{B}$ forms $\mathcal{O}_{i+1}$ by wrapping O_{i+2}^{+} and replacing the key block K_{i+1}^1 with c. $\mathcal{B}$ then wraps O_{i+1}^{+} as in Hybrid^3, again replacing the committee $\mathcal{Q}_i$'s shares, to obtain $\mathcal{O}_1$.
7. $\mathcal{B}$ gives $\mathcal{O}_1$ to $\mathcal{A}$ and gives $\mathcal{A}$ query access to $\mathsf{ProcOnion}$ on behalf of the honest parties. Let $(O_{i+1}^{-})'$ equal O_{i+1}^{-}, with K_{i+1}^1 replaced by $\mathsf{Enc}_{\mathsf{pk}_{P_{i+1}^{-}}}(00\dots0)$. If $\mathcal{A}$ queries $(O_{i+1}^{-})'$ to be processed by P_{i+1}^{-}, $\mathcal{B}$ instead runs the $\mathsf{ProcOnion}$ protocol on O_{i+1}^{-}.
8. $\mathcal{B}$ submits the guess b' returned by $\mathcal{A}$.

If m^0 is chosen, K_{i+1}^1 contains key k_{i+1}, and the onion is constructed exactly as in Hybrid^3. If m^1 is chosen, K_{i+1}^1 contains key $00\dots0$, and this version of the experiment is exactly Hybrid^4. Thus, since $\mathcal{A}$ can distinguish between Hybrid^3 and Hybrid^4, $\mathcal{B}$ can guess the challenger's chosen bit.

Remark. Changing k_{i+1} in K_{i+1}^1 does not affect O_{i+1}^{-} for the oracle access in step 7. □

A.4 $\mathsf{Hybrid}^4 \approx \mathsf{Hybrid}^5$

Hybrid^5 is identical to Hybrid^4, except that $\mathcal{O}_{i+1}$ is the output of $\mathsf{FormOnion}$ on the initial segment of the routing path, rather than a wrapping of O_{i+2}^{+}.

More precisely, let $\mathcal{O}_{i+2}$ be the onion formed with the adversary's chosen routing path after the $(i+1)^{\text{th}}$ hop; let $\mathcal{O}_{i+1}$ be the onion formed by wrapping O_{i+2} once according to the challenge parameters; let $\hat{O}_{i+1}$ be the onion formed with the adversary's chosen routing path up to the

$(i+1)^{\text{th}}$ hop. Then:

$$\begin{aligned}
O^{+}_{i+2} &= (K^1_{i+2}, \ldots, K^d_{i+2}), (H^1_{i+2}, \ldots, H^d_{i+2}), U_{i+2} \\
O^{+}_{i+1} &= (K^1_{i+1}, \{K^1_{i+2}\}_{k_{i+1}}, \ldots, \{K^{d-1}_{i+2}\}_{k_{i+1}}), \\
&\quad (H^1_{i+1}, \{H^1_{i+2}\}_{k_{i+1}}, \ldots, \{H^{d-1}_{i+2}\}_{k_{i+1}}), \{U_{i+2}\}_{k_{i+1}} \\
\hat{O}^{+}_{i+1} &= (K^1_{i+1}, \{\hat{K}^1_{i+2}\}_{k_{i+1}}, \ldots, \{\hat{K}^{d-1}_{i+2}\}_{k_{i+1}}), \\
&\quad (H^1_{i+1}, \{\hat{H}^1_{i+2}\}_{k_{i+1}}, \ldots, \{\hat{H}^{d-1}_{i+2}\}_{k_{i+1}}), \{\hat{U}_{i+2}\}_{k_{i+1}}
\end{aligned}$$

where the $\hat{K}, \hat{H}$, and $\hat{U}$ segments in $\hat{O}^{+}_{i+1}$ are as defined in the recursive definition of FormOnion for poly onion encryption. Observe that K^1_{i+1} and H^1_{i+1} are identically distributed in O^{+}_{i+1} and $\hat{O}_{i+1}$, and all blocks that change between O^{+}_{i+1} and $\hat{O}^{+}_{i+1}$ are encrypted under a PRP with key k_{i+1}.

To prove that Hybrid^4 and Hybrid^5 are indistinguishable, we use the fact that the replaced parts of the onion are all encrypted under a PRP with key k_{i+1}. Our proof is a reduction to the CCA2 security of this PRP.

Proof. Suppose $\mathcal{A}$ can distinguish between Hybrid^4 and Hybrid^5. We will construct $\mathcal{B}$ that can break the CCA2 security of our PRP.

1. $\mathcal{B}$ generates the public and private keys for all parties. $\mathcal{B}$ sends the public keys for all parties to $\mathcal{A}$.
2. $\mathcal{A}$ sends the set of corrupted parties Bad, the bulletin B, the churn schedule C, and the public keys for Bad to $\mathcal{B}$.
3. $\mathcal{B}$ gives $\mathcal{A}$ oracle access to ProcOnion on behalf of the honest parties.
4. $\mathcal{B}$ receives from $\mathcal{A}$ the challenge parameters: a path length ℓ, a message m, a routing position $i < \ell$, a round r, committees $\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell$, and candidates $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$.
5. $\mathcal{B}$ constructs $\mathcal{O}_{i+2}$ using FormOnion given the challenge parameters, as in Hybrid^4. $\mathcal{B}$ samples keys $k_1, \ldots, k_i$ for constructing the rest of the onion. $\mathcal{B}$ constructs K^1_{i+1} as in Hybrid^4, replacing k_{i+1} with $00\ldots0$. $\mathcal{B}$ constructs H^1_{i+1} as prescribed by FormOnion, as in the previous hybrids.
6. $\mathcal{B}$ chooses two series of messages

   $$\begin{aligned}
   m^0 &= K^1_{i+2}, \ldots, K^{d-1}_{i+2}, H^1_{i+2}, \ldots, H^{d-1}_{i+2}, U_{i+2} \\
   m^1 &= \hat{K}^1_{i+2}, \ldots, \hat{K}^{d-1}_{i+2}, \hat{H}^1_{i+2}, \ldots, \hat{H}^{d-1}_{i+2}, \hat{U}_{i+2}
   \end{aligned}$$

   and sends them to the challenger. Call the blocks of the challenger's chosen message K^*, H^*, U^*. $\mathcal{B}$ receives back c, the encryption of each of the messages in m^b using the challenger's key k_{i+1}:

   $$c = \{K^{*1}_{i+2}\}_{k_{i+1}}, \ldots, \{K^{*d-1}_{i+2}\}_{k_{i+1}}, \{H^{*1}_{i+2}\}_{k_{i+1}}, \ldots, \{H^{*d-1}_{i+2}\}_{k_{i+1}}, \{U^*_{i+2}\}_{k_{i+1}}$$

   $\mathcal{B}$ constructs layer $i+1$ of the challenge onion, O^*_{i+1}, using these segments:

   $$\begin{aligned}
   O^*_{i+1} = (K^1_{i+1}, &\{K^{*1}_{i+2}\}_{k_{i+1}}, \ldots, \{K^{*d-1}_{i+2}\}_{k_{i+1}}), \\
   &(H^1_{i+1}, \{H^{*1}_{i+2}\}_{k_{i+1}}, \ldots, \{H^{*d-1}_{i+2}\}_{k_{i+1}}), \{U^*_{i+2}\}_{k_{i+1}}
   \end{aligned}$$

   $\mathcal{B}$ then wraps O^*_{i+1} repeatedly to obtain the challenge onions $\mathcal{O}_1$, making sure to replace the shares of k_{i+1} for committee $\mathcal{Q}_i$ with shares of $00\ldots0$, as in Hybrid^3. $\mathcal{B}$ sends $\mathcal{O}_1$ to $\mathcal{A}$.
7. $\mathcal{B}$ gives $\mathcal{A}$ oracle access to ProcOnion on behalf of the honest parties, so that when O^*_{i+1} is queried, $\mathcal{B}$ instead runs ProcOnion on O^{+}_{i+1}. Furthermore, let $(O^{*-}_{i+1})'$ equal O^*_{i+1}, with K^{*1}_{i+1} replaced by $\mathsf{Enc}_{\mathsf{pk}_{P^-_{i+1}}}(00\ldots0)$. If $\mathcal{A}$ queries $(O^{*-}_{i+1})'$ to be processed by P^-_{i+1}, $\mathcal{B}$ instead runs the ProcOnion protocol on O^-_{i+1} .
8. $\mathcal{B}$ submits the guess b' returned by $\mathcal{A}$.

Remark. In step 7, $\mathcal{B}$ simulates $\mathsf{ProcOnion}$ on O^{-}_{i+1} and O^{+}_{i+1} by using its knowledge of the decryptions of K_{i+1}, H_{i+1}, and U_{i+1}, since it formed these blocks.

If m^0 is chosen, O^{*}_{i+1} is exactly O_{i+1}, a wrapping of O_{i+2} consistent with $\mathcal{A}$'s challenge parameters. Wrapping O_{i+1} to create O_1 gives exactly Hybrid^4. If m^1 is chosen, O^{*}_{i+1} is exactly the $(i+1)^{\text{th}}$ layer of the onion formed using the initial segment of the challenge parameters. Wrapping O^{*}_{i+1} yields a challenge onion constructed as in Hybrid^5. Therefore, if $\mathcal{A}$ can distinguish between Hybrid^4 and Hybrid^5, $\mathcal{B}$ can determine whether the challenger chose m^0 or m^1 with non-negligible probability. □

A.5 $\mathsf{Hybrid}^5 \approx \mathsf{Hybrid}^6$

Hybrid^6 is the same as Hybrid^5, except that in the key block K^1_{i+1}, k_{i+1} is changed to $00\ldots0$, the role of P_{i+1} is changed from intermediary to recipient, and the identity of P_{i+2} is changed to $\perp$.

This proof is basically the same as the proof that $\mathsf{Hybrid}^3 \approx \mathsf{Hybrid}^4$. It involves a reduction to the security of Enc, the public key encryption scheme used to encrypt K^1_{i+1}.

A.6 $\mathsf{Hybrid}^6 \approx \mathsf{Hybrid}^7$

Hybrid^7 is the same as Hybrid^6, except that the shares σ_i of all committee members $\mathcal{Q}_i$ are changed back from shares of $\mathsf{Enc}_{\mathsf{pk}_{P^{-}_{i+1}}}(00\ldots0)$ to shares of $\mathsf{Enc}_{\mathsf{pk}_{P^{-}_{i+1}}}(k_{i+1}, \text{role}(P^{-}_{i+1}), P^{+}_{i+2})$, where the the role of P^{-}_{i+1} is now the recipient.

This proof is analogous to the proof that $\mathsf{Hybrid}^3 \approx \mathsf{Hybrid}^2$. It involves a reduction to the security of the secret sharing scheme, which ensures that the secret shared message cannot be reconstructed given only the corrupted parties' shares.

A.7 $\mathsf{Hybrid}^7 \approx \mathsf{Hybrid}^8$

This proof is analogous to the proof that $\mathsf{Hybrid}^1 \approx \mathsf{Hybrid}^2$, by security of Enc.

Let $\mathcal{O}^0_{i+1}$ denote the $(i+1)^{\text{th}}$ layer of the challenge onion as constructed when $b = 0$ in $\mathsf{POSecurityGame}$; i.e., the output of $\mathsf{FormOnion}$ on the challenge paramters. Let $\mathcal{O}^1_{i+1}$ denote the $(i+1)^{\text{th}}$ layer of the challenge onion as constructed when $b = 1$ in $\mathsf{POSecurityGame}$; i.e., the output of $\mathsf{FormOnion}$ on the truncated routing path. Observe that now, in Hybrid^8, the $\mathsf{ProcOnion}$ oracle access in the second phase is defined such that if an onion in $\mathcal{O}^1_{i+1}$ is queried, the oracle instead processes the corresponding onion in $\mathcal{O}^0_{i+1}$. This oracle access is consistent with the access in $b = 1$ of $\mathsf{POSecurityGame}$.