Bruisable Onions: Anonymous Communication in the Asynchronous Model

Megumi Ando* Anna Lysyanskaya† Eli Upfal‡

June 3, 2024

Abstract

In onion routing, a message travels through the network via a series of intermediaries, wrapped in layers of encryption to make it difficult to trace. Onion routing is an attractive approach to realizing anonymous channels because it is simple and fault tolerant. Onion routing protocols provably achieving anonymity in realistic adversary models are known for the synchronous model of communication so far.

In this paper, we give the first onion routing protocol that achieves anonymity in the asynchronous model of communication. The key tool that our protocol relies on is the novel cryptographic object that we call *bruisable* onion encryption. The idea of bruisable onion encryption is that even though neither the onion's path nor its message content can be altered in transit, an intermediate router on the onion's path that observes that the onion is delayed can nevertheless slightly damage, or bruise it. An onion that is chronically delayed will have been bruised by many intermediaries on its path and become undeliverable. This prevents timing attacks and, as we show, yields a provably secure onion routing protocol in the asynchronous setting.

*Computer Science Department, Tufts University, `mando@cs.tufts.edu`

†Computer Science Department, Brown University, `anna@cs.brown.edu`

‡Computer Science Department, Brown University, `eli@cs.brown.edu`

Contents

1 Introduction

The ability to communicate anonymously is an increasingly vital component of digital life and citizenship. From Iranian protesters wishing to safely to inform the world what is happening in the streets of Tehran, to Russian citizens trying to communicate with outside media, anonymity gives people all over the world a chance to exercise their fundamental rights without fear of repercussions. Practical tools such as Tor [DMS04] (i.e., "The onion router," inspired by Chaum's onion routing idea [Cha81] described below) or VPNs have a lot of room for improvement. Both are easily blocked, and neither guarantees privacy even from the network adversary (e.g., a standard model for a resourceful ISP- or AS-level adversary) [MD05, SEV+15, WSJ+18, Rop21].

A communications protocol is anonymous [ALU21] if for any pair of input vectors (σ_0, σ_1) that differ only on the inputs and outputs[1] of honest parties (e.g., Alice sends to Bob in σ_0 and to Charlie in σ_1), the adversary (whose capabilities vary depending on the adversarial model) cannot tell from interacting with the honest nodes in a protocol run whether the input was σ_0 or σ_1.[2]

The goal of research on onion routing [Cha81, Cha88, CL05, vdHLZZ15, ALU18, KBS20, ALU21, AL21, KHRS21, ACLM22] is to achieve this definition in the presence of a malicious adversary corrupting a fraction of the participants, with a communication- and computation- efficient, fault-tolerant and decentralized protocol. In an onion routing protocol, to send a message to Bob, Alice first picks a sequence of intermediary parties $I_1, \ldots, I_{\ell-1}$ and then forms a layered cryptographic object called an onion using the message and the routing path $(I_1, \ldots, I_{\ell-1}, \mathrm{Bob})$. Alice then sends the onion to the first intermediary I_1 on the routing path who peels off just the outermost layer of the onion (i.e., processes the onion) and sends the peeled onion O_2 to the next party I_2 on the routing path, I_2 peels O_2 and sends the peeled onion O_3 to I_3, and so on. This procedure continues until Bob receives the message from Alice.

In an onion routing protocol that uses standard cryptographic onions [CL05], even a powerful adversary who can corrupt (and "look into" or even control) some of the parties cannot link an honest party's incoming onion to its outgoing onion. This lack of transparency allows for shuffling onions when they are batch-processed at an honest party [RS93, BFT04, IKK05, ALU18].

1.1 Technical challenge: asynchronous onion routing

In recent years, several protocols were presented as provably secure yet practical solutions [CBM15, vdHLZZ15, TGL+17, KCDF17, ALU18, ALU21]. However, all these protocols' security analysis requires synchronous communication. In the synchronous communications setting, time progresses in rounds, and message transmissions are lossless and instantaneous. While modeling communications in this way makes designing and analyzing anonymity protocols more tractable, it is somewhat of a cheat. Currently deployed anonymity protocols, such as Tor [DMS04] and Loopix [PHE+17], are known to be vulnerable to traffic analysis attacks [MD05, SEV+15, WSJ+18, AMWB23] that exploit the asynchronous nature of communication in the real world.

Constructing a solution for the asynchronous setting is challenging because the adversary can easily influence the traffic flow, for example, by mounting a BGP interception attack [SEV+15], so that a targeted message arrives with an expected and observable delay. (See Section 1.2 for an example of a timing attack on a preciously known solution.) The adversary can do this even if the onions are batch-processed and even if we are willing to pay a cost by increasing the latency and/or

[1] Here, by "output" of a party P we mean a set of messages $\{m\}$ such that some party P' receives (m, P) as part of its input. I.e. P' intends to send m to P.

[2] Alternative definitions of anonymity exist [BKM+13, KBS+19], but we will be referring to the standard cryptographic definition here.

volume of dummy traffic. As we explain below, this attack method breaks the anonymity of every known protocol designed and proven secure for the synchronous setting; this is a problem that is not trivially fixable by using sychronizers (which assume no failures) or clock synchronization algorithms (which guarantees that most if not all of the honest parties are synchronized) [Lyn96]. In this paper, we present the first **provably anonymous** onion routing protocol for the asynchronous communications setting.

1.2 Towards a solution: a discussion

Starting point: solution for the synchronous setting. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ be participants in an onion routing protocol. In the synchronous setting, it is possible to achieve anonymity against the passive adversary (who observes all network traffic and passively observes at a constant fraction of the parties) by thoroughly shuffling together the messages. Consider the simple protocol, Π_p. In this protocol, each participant $P \in \mathcal{P}$ receives a message-recipient pair (m, R) as input and forms a single onion using m and the routing path $(I_1, \ldots, I_{\ell-1}, R)$ where each $I_j \in \mathcal{P}$ is chosen independently and uniformly at random from a set of servers (some subset of the participants). Ando, Lysyanskaya, and Upfal showed that Π_p is anonymous so long the expected server load (the number of onions that each server processes in a round) and the round complexity are both at least polylogarithmic in the security parameter [ALU18].

However, Π_p is not anonymous against the active adversary (who controls the corrupted parties and can make them deviate from the protocol). The active adversary can direct corrupted nodes to drop onions and learn who is talking with whom by observing who receives fewer messages than anticipated. For example, if the first intermediary on the routing path from Alice to her recipient (Bob) is adversarial (which happens with constant probability), the adversary can drop Alice's onion in the first round and learn who Alice's recipient is when Bob doesn't receive a message in the end. Additionally, the adversary can direct corrupted parties to replace onions formed by honest senders with ones they generate. In such an attack, the adversary can trick the honest parties into believing that onions (sufficiently) shuffle when they don't since the adversary knows what the onions they generate look like. We can circumvent this attack using checkpoint dummy onions [ALU18,ALU21]. For cryptographic reasons explained in Preliminaries, the adversary cannot forge checkpoint onions; thus, if the adversary drops too many onions, each party independently realizes this when they observe correspondingly far fewer checkpoint onions.

A natural idea for an onion routing protocol is for each party P_i to form a random number (polylogarithmic in the security parameter) of checkpoint onions (each for a randomly chosen recipient), along with an onion bearing the actual payload for the P_i's recipient. In such a scenario, one of two things can happen. If the adversary drops many onions, then the protocol aborts when the parties detect this from the missing checkpoint onions; otherwise, the checkpoint onions provide sufficient cover for the message-bearing onions. That is, as shown by Ando, Lysyanskaya, and Upfal, this protocol (dubbed Π_a – "a" for "active adversary") is differentially private from the active adversary corrupting at most a constant fraction of the parties in the synchronous model [ALU18]. Specifically, the adversarial views corresponding to any two neighboring input vectors that differ only on honest parties' inputs and outputs, are statistically similar as defined by standard differential privacy; see Definition 1.

Defining local clocks for Π_a. To adapt Π_a for the asynchronous model, we must contend with the fact that there are no global rounds. Each party may, however, keep a local clock. Our first idea is that a participant P_i advances his clock based on some way of satisfying himself that most of the onions that meant to arrive in the current epoch (according to the local clock) have already arrived; say some τ fraction of them. We can use checkpoint onions to achieve this. Additionally, P_i sends out (processed) onions in batches only when it advances its clock. This way, these onions are guaranteed to shuffle since P_i processes onions only once a sufficient number of them have been received.

Motivation for bruisable onions. Unfortunately, this approach does not quite work. As mentioned previously, in the asynchronous setting, the main challenge is preventing the adversary from mounting a timing attack that compromises anonymity. For example, the adversary can delay one of Alice's onions but not delay or drop any other onion. Assuming that the protocol is running continuously, this will ensure that the adversary will observe a late onion delivery at Alice's recipient with (non-negligibly) higher probability than at any other recipient. So, what we want is a mechanism that drops onions that are (chronically) running behind. A first attempt at accomplishing this might be to mark a layer in the middle of each onion. E.g., if the onion O consists of layers $O = (O_1, \ldots, O_\ell)$ for the parties on the routing path $(I_1, \ldots, I_{\ell-1}, R)$, then peeling $O_{\ell/2}$ reveals that it is layer $\ell/2$. The processing party $I_{\ell/2}$ can use this information to determine whether $O_{\ell/2}$ is late relative to its local clock. The problem with this approach is that when $I_{\ell/2}$ is adversarial, $I_{\ell/2}$ may not drop $O_{\ell/2}$.

Our solution is to use cryptographic means to allow a few different intermediaries (polylogarithmic in the security parameter in number and randomly chosen) to each "bruise" an onion if it arrives late. The idea is that an onion that is chronically running behind will be bruised many times and will not reach its final destination (its recipient) because it will have accumulated too many bruises for the innermost onion to be recoverable.

Note that the parties don't immediately drop onions upon late arrivals. If they did, the protocol – even under good network conditions – would not deliver any message. This is because τ fraction of the onions arriving on time "in epoch $j-1$" doesn't translate into each party eventually receiving τ fraction of the expected j^{th} layer checkpoint onions. More likely, some parties will not receive enough checkpoint onions to progress, and the protocol will stall. So, what we want is for onions to be "bruisable;" that is, a party can "bump" an onion so that the damage to it (the "bruise") shows up only later. Within the context of our protocol, a "bruised" onion can travel on, unnoticed by others that it has been modified in any way until it reaches the last intermediary $I_{\ell-1}$ at which point the damage is finally discovered. If the damage is great enough, $I_{\ell-1}$ is unable to extract the identity of the recipient from the bruised onion.

Our contributions. Our list of contributions in this paper are as follows:

- **A new cryptographic primitive: bruisable onion encryption.** See Section 3 for the formal definition including the correctness and security properties. Other than for the application to onion routing in the asynchronous model, bruisable encryption is interesting because it is an example of an encryption scheme that is both malleable in a way that's useful in an application (since an intermediary is explicitly allowed to bruise an onion) and yet provide security against an adversary who is allowed to adaptively query participants to process onions of its choice.
- **A construction of a bruisable onion encryption scheme: Tulip Onion Encryption Scheme (TOES).** See Section 4 for the construction, and Section 4.2 for the proof of security. Specifically, we show that TOES is bruisable-onion secure (Definition 2) assuming the existence of CCA2-secure public encryption schemes with tags, block ciphers, and collision-resistant hash functions (Theorem 1).
- **The first provably anonymous onion routing protocol in the asynchronous setting:**

Π_t **("t" for "tulip" or "threshold").** See Section 5 for the construction and Section 6 for the analysis of our protocol. We show that for small constant corruption rate (e.g., 10%) and drop rate (e.g., 10%), our protocol simultaneously guarantees: a positive constant message delivery rate (Theorem 2) and $(\epsilon, \mathsf{negl}(\lambda))$-differential privacy from the active adversary for any constant $\epsilon > 0$ (Theorem 3 and Corollary 1). The anonymity guarantee holds for any corruption rate strictly less than 50% (and any drop rate). The message delivery guarantee holds even in the extreme case where the adversary chooses to bruise every onion layer it receives. In the setting where the adversary is maliciously bruising onions only at 5% of the parties and not dropping onions, the guaranteed message delivery rate is over 0.85.

2 Preliminaries

2.1 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 (abbreviated w.o.p.) if its complement occurs with negligible probability in the security parameter λ. Similar to the convention that $\mathsf{poly}(\lambda)$ means polynomially bounded in λ, we introduce an analogous notation $\mathsf{polylog}(\lambda)$, by which we means polylogarithmically bounded in λ. Throughout the paper, we use the symbol $\perp$ to indicate a dummy object (such as a dummy message or a dummy recipient).

2.2 Modeling the problem

System parameters. Let λ be the security parameter. We assume that every quantity of the system, including the number N of participants, is bounded by a polynomial in λ.

Parties. Let $\mathsf{Parties} = \{P_1, \ldots, P_N\}$ be the static set of participants. We assume a setting with a public-key infrastructure (PKI); more precisely, we assume that every participant knows the set $\mathsf{Parties}$ and the public key $\mathsf{pk}(P)$ associated with each party $P \in \mathsf{Parties}$.

Inputs. The input σ_i for each party $P_i \in \mathsf{Parties}$ is a set of message-recipient pairs, that is, $\sigma_i = \{(m_{i,1}, R_{i,1}), \ldots, (m_{i,l}, R_{i,l})\}$, where the inclusion of a message-recipient pair $(m_{i,j}, R_{i,j})$ means that P_i is instructed to send the message $m_{i,j}$ to the recipient $R_{i,j}$. By the input vector, we mean the vector $\sigma = (\sigma_1, \ldots, \sigma_N)$ containing everyone's inputs.

Two input vectors σ_0 and σ_1 are neighboring if they are the same except that the honest destinations for a pair of messages originating at honest parties are swapped. More precisely, there exist $(m, P_u) \in \sigma_{0,i}$ and $(m', P_v) \in \sigma_{0,j}$ such that $\sigma_{1,i} = (\sigma_{0,i} \cup \{(m', P_v)\}) \setminus \{(m, P_u)\}$, $\sigma_{1,j} = (\sigma_{0,j} \cup \{(m, P_u)\}) \setminus \{(m', P_v)\}$, and $\sigma_{1,k} = \sigma_{0,k}$ for all $k \in [N] \setminus \{i, j\}$.

Adversary model. The adversary is active, meaning that in addition to observing all network traffic, the adversary can also corrupt and control up to a constant χ fraction of the parties. The adversary chooses which parties to corrupt prior to the execution of the protocol. For our result on guaranteed message delivery, we further assume that the adversary may drop (at corrupted parties) up to a constant γ fraction of the honest parties' message packets.

Message schedule. The N parties form an asynchronous network, connected pairwise by authenticated channels. Every message on the channels is guaranteed eventual delivery after an arbitrarily long delay chosen by the adversary. This setting is in keeping with how the message schedule is modeled in Byzantine consensus literature [Bra84, CR93]; here, the adversary maintains a queue of messages that have yet to be delivered and decides which messages are delivered next. Combined with the adversary's power to control the corrupted parties to behave arbitrarily, this has the net effect that the adversary fixes the message schedule and additionally can add/drop messages at corrupted nodes.

Adversarial view. Given a communications protocol Π, adversary $\mathcal{A}$, and input vector σ, let $\mathsf{View}^{\Pi,\mathcal{A}}(\sigma)$ denote the adversary's view in a run of Π on input σ in the presence of the adversary $\mathcal{A}$; that is, $\mathsf{View}^{\Pi,\mathcal{A}}(\sigma)$ is a random variable representing everything that the adversary can observe including the network traffic and the states and computations of the corrupted parties.

2.3 Definition of anonymity

The notion of anonymity that we use in this paper is standard (computational) differential privacy:

Definition 1 ((ϵ, δ)-DP [DMNS06])**.** *A communication protocol Π is (ϵ, δ)-differentially private if for every adversary $\mathcal{A}$ and every pair of neighboring inputs σ_0 and σ_1 and every set $\mathcal{V}$ of adversarial views,*

$$\Pr\big[\mathsf{View}^{\Pi,\mathcal{A}}(\sigma_0) \in \mathcal{V}\big] \le e^{\epsilon} \Pr\big[\mathsf{View}^{\Pi,\mathcal{A}}(\sigma_1) \in \mathcal{V}\big] + \delta.$$

We say that Π is computationally (ϵ, δ)-differentially private [MPRV09] if the above bound holds for all polynomially bounded adversaries.

2.4 Checkpoint onions

A technical challenge in realizing anonymity from the active adversary is preventing the adversary from gleaning information by biasing the number of onions that arrive at the recipients. For example, the adversary who suspects that Alice is sending a message to Bob can try to confirm their suspicion by dropping the onion originating from Alice before it shuffles with other onions.

In prior work, Ando, Lysyanskaya, and Upfal introduced a cryptographic tool called *checkpoint onions* [ALU18] (a.k.a. dummy onions). These onions do not carry a payload; instead, their purpose is to provide cover traffic for "real" payload-carrying onions. They allow intermediary parties to locally determine if the active adversary is disrupting network traffic and causing onions to get dropped. This is accomplished as follows: Each pair of networked parties (the end-users as well as the intermediaries) (P_i, P_j) is associated with a secret key $s_{i,j}$ for a pseudorandom function $F_{s_{i,j}}$. This function mostly evaluates to something other than 0^k, but if $F_{s_{i,j}}(x) = y \neq 0^k$, then party P_i expects to receive an onion containing the string y in round r. I.e. party P_j must form an onion such that, at round r, this onion will reach party P_i and contain the string y. If party P_i is expecting such an onion but does not receive it, it means that the active adversary has disrupted the network.

3 Bruisable Onion Encryption

We introduce a new cryptographic primitive called bruisable onion encryption. Unlike in standard onion encryption, in bruisable onion encryption, each mixer on the routing path has a choice to add an extra bit of information to the onion: to ding (bruise) the onion or not. If the onion sustains too many bruises (i.e., a sufficient number of the mixers on the path bruise the onion), then the identity of the recipient R and the innermost onion O_ℓ for the recipient become unrecoverable.

Another difference between standard onion encryption and bruisable onion encryption is the addition of a new type of intermediaries, called *gatekeepers*. A bruisable onion O travels along its routing path $(M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ consisting of some ℓ_1 mixers, followed by some ℓ_2 gatekeepers and the recipient R. While the role of the mixers is to batch-process the onion (along with other onions) or to bruise it, gatekeepers are responsible for routing the onion all the way to the recipient only if the mixers didn't bruise it too much.

3.1 I/O syntax

A bruisable onion encryption scheme consists of the following algorithms:

KeyGen takes the security parameter 1^λ and the name of a party P as input, and outputs a public key pair $(\mathsf{pk}(P), \mathsf{sk}(P))$, i.e.,

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

$\mathsf{FormOnion}$ takes a (fixed length) message m, a routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ consisting of ℓ_1 "mixers" and ℓ_2 "gatekeepers," the public keys of the parties in $\vec{P}$, and a sequence $\vec{y} = (y_1, \ldots, y_{\ell_1+\ell_2})$ of metadata where the metadata string y_i is intended for the i^{th} processing party on the routing path. (The metadata conveyed to each intermediary is a useful component of an onion routing protocol: it allows the sender to communicate something about the onion to the processing party. For example, in our protocol in Section 5, the metadata is the pseudorandom nonces in the checkpoint onions.) $\mathsf{FormOnion}$ outputs a list of lists of onions $\vec{O} = (\vec{O}_1, \ldots \vec{O}_\ell)$ where $\ell = \ell_1 + \ell_2 + 1$. That is, letting $\mathsf{pk}(\vec{P})$ denote the public keys of the parties in $\vec{P}$,

$$\vec{O} = (\vec{O}_1, \ldots \vec{O}_\ell) \leftarrow \mathsf{FormOnion}(m, \vec{P}, \mathsf{pk}(\vec{P}), \vec{y}).$$

In standard onion encryption as defined by Camenisch and Lysyanskaya [CL05], $\mathsf{FormOnion}$ outputs a list of *onions*, $O = (O_1, \ldots, O_\ell)$. This list is called the "evolution of the onion" because it is how the onion should evolve as it travels along the routing path; each O_i is the onion that the i^{th} intermediary should receive and process.

In bruisable onion encryption, the evolution depends on if and when the onion gets bruised. Accordingly, $\mathsf{FormOnion}$ outputs a list of lists of onions, $(\vec{O}_1, \ldots \vec{O}_\ell)$, where each list $\vec{O}_i$ contains all possible variations of the i^{th} onion layer. The first list $\vec{O}_1 = (O_1)$ contains just the onion for the first mixer. For $2 \leq i \leq \ell_1$, the list $\vec{O}_i$ contains i options, $\vec{O}_i = (O_{i,0}, \ldots, O_{i,i-1})$; each $O_{i,j}$ is what the i^{th} onion layer should look like with j prior bruises. For $\ell_1 + 1 \leq i \leq \ell_1 + \ell_2$, the list $\vec{O}_i$ contains $\ell_1 + 1$ options, depending on the total bruising from the mixers. The last list $\vec{O}_\ell = (O_{\ell_1+\ell_2+1})$ contains just the innermost onion for the recipient.

Note that the routing path $\vec{P}$ may start and/or end with a sub-path consisting of dummy parties, in which case $\mathsf{FormOnion}$ outputs onions for only the non-dummy routing parties. For example, if the routing path is $(\perp, \perp, P_3, P_4, P_5, \perp, \ldots, \perp)$, $\mathsf{FormOnion}$ outputs $(\vec{O}_3, \vec{O}_4, \vec{O}_5)$.

$\mathsf{PeelOnion}$ takes the secret key $\mathsf{sk}(P)$ of the processing party P and an onion O. Its output is (i, y, O', P') where i is the position of the party P on the onion's routing path and y is the metadata, while (O', P') falls into one of four cases: if P is not the recipient, (O', P') is either (1) the peeled onion O' and its next destination P' or (2) $(\perp, \perp)$ if the onion is malformed or too bruised; if P is the recipient, then $P' = \perp$, while O is either (3) a message m or (4) $\perp$.

$$(i, y, O', P') \leftarrow \mathsf{PeelOnion}(\mathsf{sk}(P), O).$$

BruiseOnion is an algorithm that allows an intermediary to damage the onion, or *bruise* it. This option is only available to the mixers on the routing path, i.e., to the first ℓ_1 intermediaries. BruiseOnion takes as input the secret key $\mathsf{sk}(P)$ of the party P and the onion O to be bruised as input, and outputs a bruised onion O' to send to its next destination,

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

3.2 Correctness definition

If a bruisable onion is processed only either by running the PeelOnion algorithm or the BruiseOnion algorithm at every hop, we require that it should travel along the intended routing path specified by the sender. Further, if the bruising isn't too bad (i.e., it falls under some threshold θ), the gatekeepers should be able to recover the innermost onion and the recipient; otherwise, routing the onion through $(G_1, \ldots, G_{\ell_2})$ should reveal the empty final destination $\perp$. We formalize this intuition below.

Let $\Sigma = (\mathsf{KeyGen}, \mathsf{FormOnion}, \mathsf{PeelOnion}, \mathsf{BruiseOnion})$ be a bruisable encryption scheme.

Let Parties be any set of participants.

For each $P_i \in \mathsf{Parties}$, let $(\mathsf{pk}(P_i), \mathsf{sk}(P_i))$ be the key pair generated by running KeyGen on P_i.

Let m be any message from the message space; let $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ be any list of parties in Parties; let $\vec{y} = (y_1, \ldots, y_{\ell_1+\ell_2})$ be any sequence of metadata. Let $\ell = \ell_1 + \ell_2 + 1$. $(\vec{O}_1, \ldots, \vec{O}_\ell)$ is the result of running FormOnion on m, $\vec{P}$, the public keys $\mathsf{pk}(\vec{P})$ of the parties in $\vec{P}$, and $\vec{y}$, i.e.,

$$\vec{O} = (\vec{O}_1, \ldots \vec{O}_\ell) \leftarrow \mathsf{FormOnion}(m, \vec{P}, \mathsf{pk}(\vec{P}), \vec{y}).$$

We say that Σ is correct with respect to the threshold $0 < \theta \leq 1$, the number ℓ_1 of mixers, and the number ℓ_2 of gatekeepers if the following conditions are satisfied:

- **Correct peeling and bruising.** For $1 \leq i < \ell_1 + \ell_2$, $1 \leq j \leq |\vec{O}_i|$, let (i', y, O, P) be the output of $\mathsf{PeelOnion}(\mathsf{sk}(P_i), O_{i,j})$. Then $i' = i$, $y = y_i$, $O = O_{i+1,j}$, and $P = P_{i+1}$. In other words, when processing an onion, the mixer correctly recovers its position i in the list of processing parties, its metadata y_i, the onion $O_{i+1,j}$ to send forth with the same amount of bruising, and its destination P_{i+1}. Moreover, for $1 \leq i \leq \ell_1$, $1 \leq j \leq |\vec{O}_i|$, let O' be the output of $\mathsf{BruiseOnion}(\mathsf{sk}(P_i), O_{i,j})$. Then $O' = O_{i+1,j+1}$.
- **Correct gatekeeping.** For $i = \ell_1 + \ell_2$, $1 \leq j \leq \ell_1 + 1$, let (i', y, O, P) be the output of $\mathsf{PeelOnion}(\mathsf{sk}(P_i), O_{i,j})$. If $j \leq \theta\ell_1$, then $i' = i$, $y = y_i$, $O = O_\ell$, and $P = R$. In other words, when processing an onion that is not too bruised, the last gatekeeper correctly recovers its position $i = \ell_1 + \ell_2$ in the list of processing parties, its metadata y_i, the onion O_ℓ to send forth, and the recipient R. However, if $j > \theta\ell_1$, then $i' = i$, $y = y_i$, $O = \perp$, and $P = \perp$. In other words, if the onion is too bruised, the honest gatekeeper still recovers its metadata but not the onion to send forth or the next destination.
- **Correct message.** Peeling the innermost onion layer recovers the intended message, i.e., $\mathsf{PeelOnion}(\mathsf{sk}_R, O_\ell) = (\ell, \perp, m, \perp)$.

3.3 Security definition

We define security for bruisable onion encryption using the following game, BrOnSHH (which stands for bruisable onion security with an honest mixer and an honest gatekeeper). BrOnSHH is parameterized by the security parameter 1^λ, the adversary $\mathcal{A}$, the bruisable onion encryption scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{FormOnion}, \mathsf{PeelOnion}, \mathsf{BruiseOnion})$, and the system parameters θ (which controls how much bruising can be tolerated) and ℓ_1 and ℓ_2 (which specify the numbers of mixers and gatekeepers for an onion's path).

The challenger controls an honest mixer, an honest gatekeeper, and an honest recipient. The challenge onion might or might not be intended for the honest recipient, but it must be routed through the honest mixer and gatekeeper. The adversary controls all intermediaries other than the honest mixer, the honest gatekeeper, and the honest recipient. (As stated under in **Adversary model**, the corruptions are modeled as non-adaptive.)

- **Setup:** The adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ set up the parties' keys.
  1. The adversary $\mathcal{A}$ sends the names of the honest mixer P_m, the honest gatekeeper P_g, the honest recipient P_r, and the adversarial parties Bad; and the public keys $\mathsf{pk}(\mathsf{Bad})$ of the adversarial parties to the challenger $\mathcal{C}$.
  2. For each honest party $P \in \{P_m, P_g, P_r\}$, $\mathcal{C}$ generates a key pair $(\mathsf{pk}(P), \mathsf{sk}(P)) \leftarrow \mathsf{KeyGen}(1^\lambda, P)$ and sends $\mathsf{pk}(P_m), \mathsf{pk}(P_g), \mathsf{pk}(P_r)$ to $\mathcal{A}$.
- **First query phase:**
  3. $\mathcal{A}$ can direct an honest party to peel or bruise an onion by submitting queries to peel (resp. bruise) an onion O on behalf of an honest party $P \in \{P_m, P_g, P_r\}$, in which case $\mathcal{C}$ responds with the output of $\mathsf{PeelOnion}(\mathsf{sk}(P), O)$ (resp. $\mathsf{BruiseOnion}(\mathsf{sk}(P), O)$).
- **Challenge phase:** $\mathcal{A}$ picks the parameters of the challenge onion, and $\mathcal{C}$ replies with the challenge onion O_1.
  4. $\mathcal{A}$ sends to $\mathcal{C}$: the message m; the routing path $\vec{P}$ where $P_m = M_{i_1}$ in position $i_1 \le \ell_1$ is one of the mixers $(M_1, \ldots, M_{\ell_1})$, $P_g = G_{i_2-\ell_1}$ in position $\ell_1 < i_2 \le \ell_1 + \ell_2$ is one of the gatekeepers $(G_1, \ldots, G_{\ell_2})$, and the recipient R may be P_r; and the sequence $\vec{y} = (y_1, \ldots, y_{\ell_1+\ell_2})$ of metadata.
  5. $\mathcal{C}$ samples a bit $b \leftarrow\$ \{0, 1\}$.
     - If $b = 0$, $\vec{Q} = \vec{P}$. $\vec{z} = \vec{y}$.
     - If $b = 1$, $\vec{Q} = (M_1, \ldots, M_{i_1-1}, P_m, \overbrace{\perp, \ldots, \perp}^{\ell_1+\ell_2+1-i_1})$. $\vec{z} = (y_1, \ldots, y_{i_1}, \overbrace{\perp, \ldots, \perp}^{\ell_1+\ell_2+1-i_1})$.

     $\mathcal{C}$ returns the first onion O_1 in the output from running $\mathsf{FormOnion}$ on m, $\vec{Q}$, the public keys $\mathsf{pk}(\vec{Q})$, and $\vec{z}$, i.e., $((O_1), \vec{O}_2, \ldots, \vec{O}_{\ell_1+\ell_2+1}) \leftarrow \mathsf{FormOnion}(m, \vec{Q}, \mathsf{pk}(\vec{Q}), \vec{z})$.
- **Second query phase:** $\mathcal{A}$ is again allowed to submit queries to have an onion peeled or bruised by an honest party P.
  6. If $b = 0$; or the request *isn't*
     - to peel or bruise an onion in $\vec{O}_{i_1}$ as the mixer P_m (query type 1),
     - to peel an onion in $\vec{O}_{i_2}$ as an honest gatekeeper P_g (type 2), or
     - to peel the onion $O_{\ell_1+\ell_2+1}$ as the recipient P_r (type 3);

     the challenger processes the request by running the scheme's algorithm (as before).
  7. If the query *is* type 1, 2, or 3 (defined above), and *this is not the first request of this type*; the challenger responds with an error message.
  8. Else ($b = 1$):
     i. **Query type 1:** the query is to the mixer P_m to peel or bruise an onion $O_{i_1,j} \in \vec{O}_{i_1}$. $\mathcal{C}$ runs $\mathsf{FormOnion}$ on the dummy message $\perp$ and the path after P_m to P_g, i.e.,

$$\vec{Q}_{i_1+1\to i_2} = (\overbrace{\perp, \ldots, \perp}^{i_1}, M_{i_1+1}, \ldots, M_{\ell_1}, G_1, \ldots, G_{i_2-\ell_1-1}, P_g, \overbrace{\perp, \ldots, \perp}^{\ell_1+\ell_2+1-i_2})$$
$$\vec{z}_{i_1+1\to i_2} = (\overbrace{\perp, \ldots, \perp}^{i_1}, y_{i_1+1}, \ldots, y_{i_2}, \overbrace{\perp, \ldots, \perp}^{\ell_1+\ell_2+1-i_2})$$
$$\vec{O}_{i_1+1\to i_2} \leftarrow \mathsf{FormOnion}(\perp, \vec{Q}_{i_1+1\to i_2}, \mathsf{pk}(\vec{Q}_{i_1+1\to i_2}), \vec{z}_{i_1+1\to i_2}).$$

Suppose the query was to peel (resp. bruise); $\mathcal{C}$ sets $\mathsf{bruisecount} = j$ (resp. $\mathsf{bruisecount} = j+1$) and returns $(i_1, y_{i_1}, O_{i_1+1,0},\ M_{i_1+1})$ to $\mathcal{A}$ where $O_{i_1+1,0}$ is the first onion in the output $\vec{O}_{i_1+1\to i_2} = ((O_{i_1+1,0}, \ldots,\ O_{i_1+1,i_1}), \vec{O}_{i_1+2}, \ldots, \vec{O}_{i_2})$ of $\mathsf{FormOnion}$. ($\mathsf{bruisecount}$ is the number of bruises that the onion acquires before reaching M_{i_1}. The challenger keeps track of this information to ensure that the innermost onion is recoverable only if it should be.)

ii. **Query type 2:** the query is to the gatekeeper P_g to peel an onion $O_{i_2,j} \in \vec{O}_{i_2}$. Let $m' = m$ if $R = P_r$ or $\mathsf{bruisecount} + j \leq \theta\ell_1$; otherwise, let $m' = \perp$. Let $R' = R$ if $\mathsf{bruisecount} + j \leq \theta\ell_1$; otherwise, let $R' = \perp$. $\mathcal{C}$ runs $\mathsf{FormOnion}$ on the message m' and the routing path consisting of the gatekeepers after P_g and the recipient R', i.e.,

$$\begin{aligned}
\vec{Q}_{i_2+1\to} &= (\overbrace{\perp, \ldots, \perp}^{i_2}, G_{i_2-\ell_1+1}, \ldots, G_{\ell_2}, R') \\
\vec{z}_{i_2+1\to} &= (\overbrace{\perp, \ldots, \perp}^{i_2}, y_{i_2+1}, \ldots, y_{\ell_1+\ell_2}) \\
\vec{O}_{i_2+1\to} &\leftarrow \mathsf{FormOnion}(m', \vec{Q}_{i_2+1\to}, \mathsf{pk}(\vec{Q}_{i_2+1\to}), \vec{z}_{i_2+1\to})
\end{aligned}$$

$\mathcal{C}$ returns $(i_2, y_{i_2}, O_{i_2+1,0}, M_{i_2+1})$ to $\mathcal{A}$ where $O_{i_2+1,0}$ is the first onion in the output $\vec{O}_{i_2+1\to} = ((O_{i_2+1,0}, \ldots, O_{i_2+1,i_1}), \vec{O}_{i_2+2}, \ldots \vec{O}_{\ell_1+\ell_2+1})$ of $\mathsf{FormOnion}$.

iii. **Query type 3:** the query is to the recipient P_r to peel the onion $O_{\ell_1+\ell_2+1}$. $\mathcal{C}$ returns the message m.

- At the end, $\mathcal{A}$ outputs a guess b' for the bit b and wins if $b' = b$.

We define bruisable-onion security as follows.

Definition 2. *A bruisable onion encryption scheme Σ is bruisable-onion secure for parameters θ, ℓ_1, ℓ_2 if there exists a negligible function $\nu : \mathbb{N} \mapsto \mathbb{N}$ such that every p.p.t. adversary $\mathcal{A}$ wins the game* $\mathsf{BrOnSHH}(1^\lambda, \mathcal{A}, \Sigma, \theta, \ell_1, \ell_2)$ *with advantage at most $\nu(\lambda)$, i.e.,*

$$\left| \Pr\Big[\mathcal{A} \textit{ wins } \mathsf{BrOnSHH}(1^\lambda, \mathcal{A}, \Sigma, \theta, \ell_1, \ell_2)\Big] - \frac{1}{2} \right| \leq \nu(\lambda).$$

3.3.1 Intuition for the security definition

Our definition captures the idea that if the onion encryption scheme is secure, the adversary cannot determine any meaningful information about an onion that is "hidden behind an honest party:"

- **Layers for parties up to the honest mixer.** The adversary cannot distinguish between the scenario where the challenger forms O_1 as specified by the adversary (case $b = 0$) from the scenario where the challenger forms O_1 without using the message m, the routing path after P_m, or the metadata corresponding to the path after P_m (case $b = 1$). See step 5 of the security game.
- **Layers for parties after the honest mixer up to the honest gatekeeper.** The adversary cannot tell whether the peeled (resp. bruised) version O' of the challenge onion $O_{i_1,j}$ for P_m is obtained by peeling (resp. bruising) $O_{i_1,j}$ as specified by the adversary (case $b = 0$), or if O' is a fresh onion formed information-theoretically independently of the message m, the path and metadata up to P_m, the path and metadata after P_g, or the amount of bruising that the onion has incurred so far (case $b = 1$). See step 6 and step 8i of the security game.
- **Layers for the parties after the honest gatekeeper.** If the challenge onion incurs more than (resp. at most) the threshold number ($\theta\ell_1$) of bruises, then the innermost onion and the recipient are unrecoverable (resp. remain recoverable). In this event, the adversary cannot tell

whether the onion O'' that the gatekeeper P_g produces as the peeled version of its challenge onion $O_{i_2,j}$ was obtained by peeling $O_{i_2,j}$ (case $b = 0$), or if O' is a fresh onion information-theoretically independent of the bruising so far and, if $R = P_r$, the message m (resp. the message m, the recipient R, and the bruising so far). See step 6 and step 8ii of the security game.

- **Replay attacks.** Note that in both the real world and in our security game, the adversary can send an onion for processing to the same honest party more than once. A feature of bruisable onions is that the adversary can send different versions of the same onion, corresponding to different amounts of bruising. We cannot guarantee security if more than one version is processed according to the protocol since that would reveal how bruised an onion was when it got to the adversary. Thus, in our security game, the challenger will not process the same onion more than once, and this includes differently bruised versions of the same onion. An honest participant in a protocol that uses bruisable onion encryption needs to keep state information and do the same. It is important that a replayed onion be detectable even if it's a different version. In our construction in Section 5, different versions of the same layer of the same onion share a symmetric key; storing this key would enable one to identify and reject replayed onions.

Remark. Although we do not provide a UC functionality for bruisable onion encryption in this paper, we note that our definition here should be sufficient to UC-realize any reasonable modeling of such a functionality in the spirit of the ideal functionalities for (regular, non-bruisable) onion encryption of prior work [CL05, AL21]. In this approach, an ideal functionality for bruisable onion encryption would form onions on behalf of honest parties piece-wise. Given a routing path P, the "segments of P" are the subpaths of P that partition P in such a way that each subpath forms a contiguous sequence of adversarial parties followed by a single honest party (or no honest party if the segment is that last subpath containing an adversarial recipient). For example, letting capitalized parties denote honest parties, for the path $P = (P_1, p_2, p_3, P_4, P_5, p_6, P_7)$, the segments are (P_1), (p_2, p_3, P_4), (P_5), and (p_6, P_7). The ideal functionality forms the onion layers for each segment separately, without knowledge of the rest of the path, the message, or the bruise count so far; this ensures that onion layers across different segments are information-theoretically unrelated to each other. For each $\mathsf{FormOnion}$ query, the ideal functionality keeps track of which onion layers are part of the onion via an internal table or dictionary, as well as the cumulative bruise count. Our security definition would ensure that, whether the onion layers are formed correctly (as in the real world) or piecemeal by the simulator, no adversary can distinguish; the proof that a bruisable onion encryption scheme satisfying our definition would UC-realize such a functionality would follow the outline of the proof of Ando and Lysyanskaya [AL21], adjusted for the addition of bruises and gatekeepers. Seen in this way, the UC composition theorem allows us to analyze the anonymity of our onion routing protocol separately from the security of the onions.

4 Tulip Onion Encryption Scheme

Our onion encryption scheme produces a type of onion that we call tulip bulbs. A tulip bulb consists of three components: the header H containing the routing information, the content C containing the payload, and the "sepal" S for peeling the penultimate onion layer. (The sepal is the outermost part of a flower that protects the flower while it is still a bud. In our construction, the sepal protects the rest of the content by "absorbing" the bruising.)

Below, we explain on a high level how a tulip bulb is formed and how it will be processed;

this will be helpful for understanding our overall construction. Given a party P, let $(\mathsf{pk}(P), \mathsf{sk}(P))$ denote the public key and secret key of P; and let O_i denote the tulip bulb (one of the $|\vec{O}_i|$ options) for the i^{th} party on the routing path $(M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ of length $\ell = \ell_1 + \ell_2 + 1$.

The header H_i. In our construction, for each i, all variations in $\vec{O}_i$ have the same header H_i and content C_i; the only differences are in the sepals. The header $H_i = (E_i, B_i)$ consists of the ciphertext E_i and the rest of the header B_i. E_i is an encryption under $\mathsf{pk}(P_i)$ of the tuple (i, y_i, k_i) where i is the position, y_i is the metadata, and k_i is the layer key. For $1 \leq i < \ell - 1$, B_i is an encryption under k_i of the identity of the next processor P_{i+1} and header H_{i+1} of the tulip bulb O_{i+1} that will be sent to P_{i+1}.

The header $H_{\ell-1}$ for the last gatekeeper G_{ℓ_2} is somewhat different. The ciphertext $E_{\ell-1}$ decrypts to the key $k_{\ell-1}$; using $k_{\ell-1}$, G_{ℓ_2} can process the sepal. If the sepal is not too damaged and processing it yields the bulb master key K, then the rest of the header $B_{\ell-1}$ can be decrypted under K, yielding the identity of the recipient R and the header H_ℓ for R.

The content C_i. The content C_i is an encryption under the layer key k_i of the content C_{i+1} of O_{i+1}. If P_i is the recipient, then it is an encryption of the message.

The sepal S_i. The sepal S_i looks different depending on whether the processor P_i is a mixer M_i or a gatekeeper G_j for $j = i - \ell_1$. Specifically:

- For $1 \leq i \leq \ell_1$, the processor P_i is the mixer M_i. The sepal S_i received by M_i consists of $\ell_1 - i + 2$ blocks, $(S_{i,1}, \ldots, S_{i,\ell_1-i+2})$. For example, if $\ell_1 = 3$, then the first mixer's tulip bulb has four sepal blocks, the second mixer has three, and the last mixer receives a bulb with only two sepal blocks.

  Suppose we want a bulb to be irrevocably lost after d bruises, but $d - 1$ bruises are tolerated.[3] For the first mixer M_1, the first d sepal blocks are encryptions of the bulb master key K, salted and wrapped in layers of symmetric encryption keyed by $k_1, \ldots, k_{\ell-1}$. The rest of the sepal blocks are salted encryptions of 0 (dummies), also salted and wrapped in layers of symmetric encryption keyed by $k_1, \ldots, k_{\ell-1}$. Let $S_{1,1}, \ldots, S_{i,\ell_1+1}$ denote these sepal blocks. I.e., letting "$\langle K \rangle$" denote a sepal block that contains the bulb master key, and "$\langle 0 \rangle$," a dummy block,

  $$S_1 = (S_{1,1}, \ldots, S_{1,\ell_1+1}) = (\overbrace{\langle K \rangle, \ldots, \langle K \rangle}^{d \text{ times}}, \overbrace{\langle 0 \rangle, \ldots, \langle 0 \rangle}^{\ell_1 - d + 1 \text{ times}})$$

  To process the tulip bulb without bruising it, M_1 peels a layer of encryption from all the blocks in S_1 and then "drops" the first block from the right. So the sepal S_2 for the next processing party retains the same number of blocks with the bulb master key, i.e.,

  $$\text{unbruised } S_2^{(1)} = (S_{2,1}, \ldots, S_{2,\ell_1}) = (\overbrace{\langle K \rangle, \ldots, \langle K \rangle}^{d \text{ times}}, \overbrace{\langle 0 \rangle, \ldots, \langle 0 \rangle}^{\ell_1 - d \text{ times}})$$

  To bruise the tulip bulb, M_1 forms S_2 by dropping the first block from the *left* instead, i.e,

  $$\text{bruised } S_2^{(2)} = (S_{2,1}, \ldots, S_{2,\ell_1}) = (\overbrace{\langle K \rangle, \ldots, \langle K \rangle}^{d-1 \text{ times}}, \overbrace{\langle 0 \rangle, \ldots, \langle 0 \rangle}^{\ell_1 - d + 1 \text{ times}})$$

[3] In our onion routing protocol in Section 5, d is set so that the innermost tulip bulb is recoverable when $\leq \theta$ fraction of the bruisable layers are bruised, i.e., $d = \theta \ell_1$.

In general, to peel the sepal $S_i = (S_{i,1}, \ldots, S_{i,\ell_1-i+2})$ without bruising it, the i^{th} mixer M_i drops the rightmost sepal block S_{i,ℓ_1-i+2}. To bruise the sepal, M_i drops the leftmost sepal block $S_{i,1}$. Carrying out this procedure ensures that the only remaining sepal block in S_{ℓ_1+1} for the last gatekeeper G_1 contains the bulb master key K if and only if the number of times that the sepal was bruised is at most $d-1$. So, the i options for the sepal S_i correspond to the i distinct $\max(\ell_1 + 2 - i, 1)$ contiguous blocks (with the appropriate number of encryptions peeled off) from $\overbrace{\langle K\rangle, \ldots, \langle K\rangle}^{d \text{ times}}, \overbrace{\langle 0\rangle, \ldots, \langle 0\rangle}^{\ell_1-d+1 \text{ times}}$.

Note that if the mixer is not honest, they can rearrange the blocks or modify the sepal in an "illegal" way outside the prescribed procedures outlined above. Verification hashes are stored in the header of the tulip bulb to allow honest parties to detect when this happens. Care must be taken that these verification hashes not reveal anything about the possible sepals other than their validity. See the remark below. Moreover, if the last few mixers on the routing path are all adversarial, the adversary can attempt to "open" more than one sepal block, which could potentially leak some information about prior bruisings. The honest gatekeepers prevent this from happening since honest parties will process a tulip bulb only once, and a tulip bulb with a repeating key k_i will be treated as a different variant of the same tulip bulb. See Section 4.1 for how the sepal blocks and the verification hashes are formed.

- For $\ell_1 + 1 \leq i < \ell$, the processor P_i is the gatekeeper $G_{i-\ell_1}$. The sepal S_i received by $G_{i-\ell_1}$ is either the encryption of the bulb master key K under symmetric keys $k_i, \ldots, k_\ell$ (if the tulip bulb wasn't bruised too much), or the encryption of 0 (if it was). P_i processes the sepal by peeling a layer of encryption: S_{i+1} is the decryption of S_i under k_i.
- The last gatekeeper G_{ℓ_2} either recovers the master key K from S_ℓ or discovers that it cannot be recovered. If K is recovered, then G_{ℓ_2} can process the rest of the header H_ℓ.

Remark on how to incorporate verification hashes. Mixer M_i receives onion $O_i = (H_i, C_i, S_i)$, where S_i is one of i sepal candidates $S_i^{(1)}, \ldots, S_i^{(i)}$, as described above. In order to ensure that the sepal does not get corrupted in transit but in fact corresponds to the sepal prepared by the sender, our construction includes (in lexicographic order) the values $\{h(S_i^{(j)})\}_{1\leq j\leq i}$ for a collision-resistant hash function h. Let us go over what can go wrong with if we include only these hashes and how to fix it.

First, note that a collision-resistant hash function may still leak information about its pre-image. In a contrived example, $h(S_i^{(j)})$ may leak the position of the first occurrence of the binary string "tulip fever" in its pre-image, if any, and still remain a collision-resistant hash function. Recall that $S_i^{(j+1)}$ is obtained by dropping the first λ bits of $S_i^{(j)}$ and concatenating some additional random bits to the end; in the event that $S_i^{(j)}$ contains the string "tulip fever" in position p, $S_i^{(j+1)}$ will contain "tulip fever" in position $p - \lambda$. Thus, our contrived hash function would leak that $S_i^{(j+1)}$ is the sepal for the onion that has one additional bruising compared to one with sepal $S_i^{(j)}$.

Luckily, we show that this is not a problem if we also include additional dummy hashes of strings that could never be proper sepals. The idea is to create one random, dummy sepal block that is never included in any sepals, but that will be hashed with valid sepal blocks in a circular manner. See Formal description below.

4.1 Formal description

Our onion encryption scheme, Tulip Onion Encryption Scheme (TOES), builds on standard cryptographic primitives: a CCA2-secure public key encryption scheme with tags $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$,[4] a block cipher, and a collision-resistant hash function h. In the description below, let "$\{\cdot\}_k$" denote symmetric encryption under the key k, and let "$\}\cdot\{_k$" denote symmetric decryption under k. This notation is consistent with prior work on onion encryption schemes, namely [CL05, AL21, ACLM22].

The onion encryption scheme's key generation algorithm is just the public key encryption scheme's key generation algorithm KeyGen. We assume a public key infrastructure where the keys (for at least the honest parties) are supplied by running KeyGen. For each party P, let $(\mathsf{pk}(P), \mathsf{sk}(P))$ denote the public-key and secret-key for party P.

Below, we describe how to form a tulip bulb containing the message m for the routing path $(M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$, and the sequence $\vec{y} = (y_1, \ldots, y_{\ell-1})$ of metadata.

Generating the tulip keys. To begin with, we pick layer keys $k_1, \ldots, k_\ell$ and master key K independently and uniformly at random from the key space of our symmetric encryption scheme. As explained earlier, each k_i will be used to encrypt the onion layer for the i^{th} processing party on the routing path; the master key is needed to recover the eventual recipient.

Forming the first sepal S_1. We describe how to compute the sepal portion of the tulip bulb O_1 for the first mixer M_1 on the routing path. The sepal S_1 consists of d key-blocks (the $\langle K \rangle$-blocks) $S_{1,1}, \ldots, S_{1,d}$, as well as $\ell_1 - d + 1$ null-blocks (the $\langle 0 \rangle$-blocks) $S_{1,d+1}, \ldots, S_{1,\ell_1+1}$.

Each key-block $\langle K \rangle$ is the bulb master key K, salted and encrypted under $k_1, \ldots, k_{\ell-1}$; that is,

$$S_{1,j} = \{\ldots\{K, s_j\}_{k_{\ell-1}} \ldots\}_{k_1} \qquad \forall 1 \le j \le d$$

where $s_j \leftarrow\$ \mathsf{SaltSpace}$ is a random value from an appropriately large salt space. The procedure for forming a null-block $\langle 0 \rangle$ is essentially the same except that we wrap 0 instead of the value K in layers of encryption, i.e.,

$$S_{1,j} = \{\ldots\{0, s_j\}_{k_{\ell_1+1}} \ldots\}_{k_1} \qquad \forall d+1 \le j \le \ell_1 + 1.$$

If the sepal S_i was not processed correctly (i.e., not just peeled or bruised), then the processing party P_i should be able to detect this. To that end, we store verification hashes (i.e., hashes of all possible values that a correctly formed and processed S_i can take on, plus a few dummy hash values), denoted by $\vec{A}_i$, in the header. These hash values are computed as follows: First, let $T_{i,j}$ denote the sepal block $S_{1,j}$ without the $i-1$ outermost encryption layers, and let T_{i,ℓ_1+2} be a dummy sepal block (i.e., a truly random string of length the number of bits in a sepal block, wrapped in layers of encryptions keyed by $k_i, \ldots, k_{\ell-1}$), which we will call the "clasp" for reasons that will become evident in the next sentence. Each hash value $A_{i,j}$ is the hash of one of the $l = \max(1, \ell_1 + 2 - i)$ contiguous blocks on the ring (really a "bracelet") $(T_{i,1}, \ldots, T_{i,\ell_1+2})$ where the block after the clasp T_{i,ℓ_1+2} is $T_{i,1}$. Letting $A'_i = \{A_{i,0}, A_{i,1}, \ldots, A_{i,\ell_1+2}\}$, $\vec{A}_i$ is the vector, *sorted in lexicographic order*, of the hashes of the elements of A'_i, i.e., letting $T_{\ell-1,\ell_1+2} \leftarrow\$ \{0,1\}^{|S_{1,1}|}$,

$$\begin{aligned}
T_{i,j} &= \}\ldots\}S_{1,j}\{_{k_1}\ldots\{_{k_{i-1}} && \forall j \in [\ell_1 + 1] \\
T_{i,\ell+2} &= \{\ldots\{T_{i,\ell_1+2}\}_{k_{\ell-1}} \ldots\}_{k_i} \\
A_{i,j} &= h\left(T_{i,(j \mod \ell_1+2)}, \ldots, T_{i,(j+l-1 \mod \ell_1+2)}\right) && \forall j \in [\ell_1 + 2] \\
\vec{A}_i &= \mathsf{Sort}(\{A_{i,0}, A_{i,1}, \ldots, A_{i,\min(i,\ell_1+1)}\})
\end{aligned}$$

[4]See [CS98] for the original formal description of encryption with tags.

Note that computing the hashes can be accomplished efficiently as the number $|\vec{A_i}|$ of hashes in each onion layer is $\ell_1 + 2$. See the next section for details on where the hashes are stored. The hash values constitute all possible ranges on the bracelet; this prevents the adversarial intermediary (a mixer or a gatekeeper prior to the last gatekeeper) from learning any information about how bruised the onion is so far. The clasp (and resulting dummy values) are needed to enable detection of any illegal rearrangement of the sepal blocks.

Forming the header and content for the last onion layer. After forming the sepal S_1, we obtain the header H_1 and content C_1 via a recursive process. First, we form the last onion layer for the ℓ^{th} party (the recipient R). The content C_ℓ is just the encryption of the message m under the key k_ℓ, i.e., $C_\ell = \{m\}_{k_\ell}$. We form the tag t_ℓ by taking the hash of C_ℓ, i.e., $t_\ell = h(C_\ell)$. The tag ensures that R can peel the last layer only if they receive an onion with the correct header and content. The header H_ℓ is completed by taking the encryption under the key $\mathsf{pk}(R)$ of the role "$\mathsf{Recipient}$," the hop-index $\ell + 1$, and the key k_ℓ, i.e.,

$$\begin{aligned} E_\ell &= \mathsf{Enc}(\mathsf{pk}(R), t_\ell, (\mathsf{Recipient}, \ell, k_\ell)) \\ H_\ell &= E_\ell \end{aligned}$$

Forming the the header and content for penultimate onion layer. Next, we form the penultimate layer $H_{\ell-1}, C_{\ell-1}$ for the last gatekeeper G_{ℓ_2}. The content $C_{\ell-1}$ is the encryption of C_ℓ under the master key K, i.e.,

$$C_{\ell-1} = \{C_\ell\}_K = \{\{m\}_{k_\ell}\}_K$$

Block $B_{\ell-1,1}$ is the encryption of E_ℓ and the identity of the recipient R under the master key K, i.e.,

$$B_{\ell-1,1} = \{R, E_\ell\}_K$$

The header $H_{\ell-1}$ consists of blocks $E_{\ell-1}, B_{\ell-1,1}$ where $E_{\ell-1}$ is the encryption under the public key $\mathsf{pk}(G_{\ell_2})$ and the appropriate tag $t_{\ell-1}$ of the role "$\mathsf{LastGatekeeper}$," the hop-index $\ell - 1$, the nonce $y_{\ell-1}$, the verification hashes $\vec{A}_{\ell-1}$, and the sepal layer key $k_{\ell-1}$, i.e.,

$$\begin{aligned} t_{\ell-1} &= h(B_{\ell-1,1}, \ldots, B_{\ell-1,\ell-1}, C_{\ell-1}) \\ E_{\ell-1} &= \mathsf{Enc}(\mathsf{pk}(G_{\ell_2}), t_{\ell-1}, (\mathsf{LastGatekeeper}, \ell - 1, y_{\ell-1}, \vec{A}_{\ell-1}, k_{\ell-1})) \\ H_{\ell-1} &= (E_{\ell-1}, B_{\ell-1,1}, \ldots, B_{\ell-1,\ell-1}) \end{aligned}$$

Forming the outer layers. For $1 \le i \le \ell - 2$, the header and content H_i, C_i builds on the header and content of the previous layer H_{i+1}, C_{i+1}, similar to how the penultimate layer builds on the last layer. Here, E_i is the encryption of the processing party's role (either "Mixer" or "$\mathsf{Gatekeeper}$"), the hop-index i, the nonce y_i, the verification hashes $\vec{A_i}$, and the key k_i. See below:

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

Letting I_{i+1} be the $i + 1^{th}$ party on the path,

$$\begin{aligned} B_{i,1} &= \{I_{i+1}, E_{i+1}\}_{k_i} \\ B_{i,j} &= \{B_{i+1,j-1}\}_{k_i} & \forall 2 \le j \le \ell - j + 1 \\ t_i &= h(B_{i,1}, \ldots, B_{i,\ell-1}, C_i) \\ E_i &= \mathsf{Enc}(\mathsf{pk}(P_i), t_i, (\mathsf{Role}, i, y_i, \vec{A_i}, k_i)) \\ H_i &= (E_i, B_{i,1}, \ldots, B_{i,\ell-1}) \end{aligned}$$

See Figure 1 below for a pictorial description of the tulip bulb O_1.

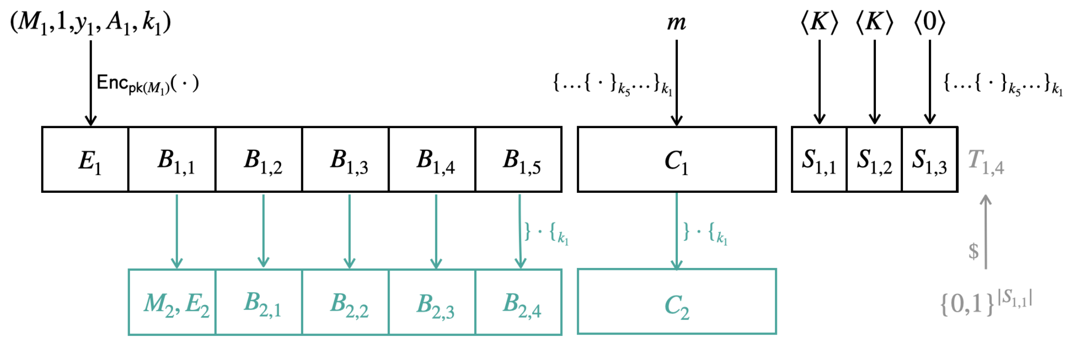

Figure 1: A pictorial description of how an onion is formed using TOES. The verification hashes A_1 for M_1 are the hashes $h(S_{1,1}, S_{1,2}, S_{1,3})$, $h(S_{1,2}, S_{1,3}, T_{1,4})$, $h(S_{1,3}, T_{1,4}, S_{1,1})$, and $h(T_{1,4}, S_{1,1}, S_{1,2})$ in lexicographical order, where $T_{1,4} \leftarrow\!\!\$\ \{0,1\}^{|S_{1,1}|}$.

Forming an onion with an incomplete path. We form an onion for a path that begins and/or ends with the empty path, e.g., $(\perp, \perp, P_3, P_4, P_5, \perp, \ldots, \perp)$, by setting the intermediary party for the empty locations (the $\perp$'s) to be the sender; and if the recipient is $\perp$, the sepal blocks are all dummy sepal blocks $\langle 0\rangle$. In this case, the algorithm outputs only the onion vectors for the parties corresponding to non-empty locations on the path.

Remark on the onion size. Recall that ℓ_1 is the number of mixers on a routing path, and ℓ_2 is the number of gatekeepers. Each onion consists of a content block, a number of sepal blocks, and a number of header blocks. The length of each message block is just the length of a message (let us call this ℓ_m). Each onion layer consists of at most $\ell_1 + 1$ sepal blocks and $\ell_1 + \ell_2 + 1$ header blocks, where the length of each sepal block is the length of each layer key (so, roughly λ), and the length of each header block is dominated by the size of the verification hashes in a layer (so, roughly $\mathcal{O}(\lambda\ell_1)$). Thus, the overall size of a tulip bulb is $\mathcal{O}\big(\lambda(\ell_1^2 + \ell_2)\big) + \ell_m$.

4.2 Proof of security

Here, we summarize our proof that our construction satisfies the definition of security provided in Definition 2.

Theorem 1. *Tulip Onion Encryption Scheme is bruisable-onion secure, assuming the existence of CCA2-secure public key encryption schemes with tags, block ciphers, and collision-resistant hash functions.*

Proof idea. We first provide a hybrid argument for the case where the challenge onion is too bruised to recover the innermost onion. Proofs for the other cases (when the onion is recoverable and/or when the recipient is honest) are given after the proof of this first case.

Below, we describe a sequence of hybrid experiments $\mathsf{Hybrid}_0, \ldots, \mathsf{Hybrid}_{18}$ and provide a brief explanation (in color) of why each pair of consecutive experiments consists of indistinguishable scenarios. (More details on the proofs of indistinguishability can be founds in Appendix A.) Recall that in the security game $\mathsf{BrOnSHH}$, the honest mixer is M_{i_1}, sitting in position $1 \le i_1 \le l_1$; and the honest gatekeeper is $G_{j=i_2-\ell_1}$, sitting in position $\ell_1 + 1 \le i_2 \le \ell_2$.

Hybrid_0: the challenge onion O_1 is formed correctly. (This is the same as the game when $b = 0$.)
$\quad\updownarrow$ Indistinguishable from CCA2-secure public key encryption.

Hybrid_1: same as Hybrid_0 except that the ciphertext E_{i_1} is an encryption under $\mathsf{pk}(M_{i_1})$ of the dummy key 0. (The challenger still samples for the layer key k_{i_1} and uses it to form the i_1^{th} onion layers $\vec{O}_{i_1}$.)

$\updownarrow$ Indistinguishable from the collision resistance of the hash function

Hybrid_2: same as Hybrid_1 except that if, in the second query phase, the challenger receives an onion $O = ((E, B), C, S) \notin \vec{O}_{i_1}$ such that $E = E_{i_1}$, the challenger responds with $\perp$ (rather than processing O).

$\updownarrow$ Indistinguishable from PRP security.

Hybrid_3: same as Hybrid_2 except that the challenger forms the i_1^{th} onion layers $\vec{O}_{i_1}$ using a truly random permutation rather than a PRP keyed with k_{i_1}.

$\updownarrow$ Identically distributed since $\vec{O}_{i_1-1}, \ldots, \vec{O}_1$ are wrapped around truly random blocks.

Hybrid_4: same as Hybrid_3 except that the challenger uses the dummy message content $\perp$ and the truncated path $(M_1, \ldots, M_{i_1}, \vec{\perp})$ and associated sequence $(y_1, \ldots, y_{i_1}, \vec{\perp})$ of metadata (instead of the real message and full path and sequence of metadata) to form O_1.

$\updownarrow$ Identically distributed since the inner layers $\vec{O}_\ell, \ldots, \vec{O}_{i_1+1}$ are independent of the path up to M_{i_1}.

Hybrid_5: same as Hybrid_4 except that the first query to peel or bruise an onion $O_{i_1} = O_{i_1,k} \in \vec{O}_{i_1}$ on behalf of M_{i_1} peels to a new onion formed using the message m, the routing path $(\vec{\perp}, M_{i_1+1}, \ldots, R)$, and the associated sequence of metadata $(\vec{\perp}, y_{i_1+1}, \ldots, y_{\ell-1})$. (The newly formed onion O_{i_1+1} has the correct number k of bruises.)

$\updownarrow$ Indistinguishable from PRP security.

Hybrid_6: same as Hybrid_5 except that the challenger forms the i_1^{th} onion layers $\vec{O}_{i_1}$ using the PRP keyed with k_{i_1} instead of a truly random permutation.

$\updownarrow$ Indistinguishable from CCA2-security.

Hybrid_7: same as Hybrid_6 except that the ciphertext E_{i_1} is an encryption of the real key k_{i_1} rather than the dummy key 0. (At this stage, the challenge onion O_1 is same as that in the game when $b = 1$, but the onions returned by M_{i_1} and G_j are not quite the same as when $b = 1$. The challenger forms O_1 using the message $\perp$, the routing path $(M_1, \ldots, M_{i_1}, \vec{\perp})$, and the metadata $(y_1, \ldots, y_{i_1}, \vec{\perp})$. The onion O_{i_1+1} returned by the challenger on behalf of the mixer M_{i_1} is a new onion with the correct number of bruises, formed by running $\mathsf{FormOnion}$ on m, $(\vec{\perp}, M_{i_1+1}, \ldots, R)$, and the metadata $(\vec{\perp}, y_{i_1+1}, \ldots, y_{\ell-1})$. The challenger obtains the onion O_{i_2+1} returned by the on behalf of the gatekeeper G_j by running $\mathsf{PeelOnion}$ on the query onion.)

$\updownarrow$ Indistinguishable from CCA2-secure public key encryption.

Hybrid_8: same as Hybrid_7 except that the ciphertext E_{i_2} is an encryption under $\mathsf{pk}(G_j)$ of the dummy key 0. (The challenger still samples for the layer key k_{i_2} and uses it to form the i_2^{th} onion layers $\vec{O}_{i_2}$.)

$\updownarrow$ Indistinguishable from the collision resistance of the hash function

Hybrid_9: same as Hybrid_8 except that if, in the second query phase, the challenger receives an onion $O = ((E, B), C, S) \notin \vec{O}_{i_2}$ such that $E = E_{i_2}$, the challenger responds with $\perp$ (rather than processing O).

$\updownarrow$ Indistinguishable from PRP security.

Hybrid_{10}: same as Hybrid_9 except that the challenger forms the i_2^{th} onion layers $\vec{O}_{i_2}$ using a truly random permutation rather than a PRP keyed with k_{i_2}.

$\updownarrow$ Identically distributed since $\vec{O}_{i_2-1}, \ldots, \vec{O}_{i_1+1}$ are wrapped around truly random blocks.

Hybrid_{11}: same as Hybrid_{10} except that the challenger uses the path $(\vec{\perp}, M_{i_1+1}, \ldots, G_j, \vec{\perp})$ and associated sequence of metadata (instead of the real message and full path and sequence of metadata) to form O_{i_1+1}.

$\updownarrow$ Identically distributed since the inner layers $\vec{O}_\ell, \ldots, \vec{O}_{i_2+1}$ are independent of the path up to G_j.

Hybrid_{12}: same as Hybrid_{11} except that the first query to peel or bruise an onion $O_{i_2} = O_{i_2,k} \in \vec{O}_{i_2}$ on behalf of G_j peels to a new onion formed using the message m, the routing path $(\vec{\perp}, G_{j+1}, \ldots, R)$, and the associated sequence of metadata. (The newly formed onion O_{i_2+1} has the correct number k of bruises.)

$\updownarrow$ Indistinguishable from PRP security.

Hybrid_{13}: same as Hybrid_{12} except that the challenger forms the i_2^{th} onion layers $\vec{O}_{i_2}$ using the PRP keyed with k_{i_2} instead of a truly random permutation.

$\updownarrow$ Indistinguishable from CCA2-security.

Hybrid_{14}: same as Hybrid_{13} except that the ciphertext E_{i_2} is an encryption of the real key k_{i_2} rather than the dummy key 0.

$\updownarrow$ Identically distributed since the sepals are truly random blocks wrapped in layers on encryption, and the verification hashes don't reveal how bruised the sepals are.

Hybrid_{15}: same as Hybrid_{14} except for how bruised the onion O_{i_1+1} is. The onion O_{i_1+1} that M_{i_1} returns will be completely unbruised. The challenger remembers how bruised O_{i_1} was, however, and forms the the onion O_{i_2+1} accordingly; thus, O_{i_2+1} is formed identically as in Hybrid_{11}. (At this stage, the challenge onions O_1 and O_{i_1+1} are the same as that in the game when $b = 1$, but the onion returned by G_j is not quite the same as when $b = 1$.)

$\updownarrow$ Indistinguishable from PRP security.

Hybrid_{16}: same as Hybrid_{15} except that the challenger forms the penultimate onion layers $\vec{O}_{\ell-1}$ using a truly random permutation rather than a PRP keyed with $k_{\ell-1}$.

$\updownarrow$ Identically distributed because since $\vec{O}_{\ell-2}, \ldots, \vec{O}_{i_2+1}$ are wrapped truly random blocks.

Hybrid_{17}: same as Hybrid_{16} except that the challenger uses the message $\perp$ and the recipient $\perp$.

$\updownarrow$ Indistinguishable from PRP security.

Hybrid_{18}: same as Hybrid_{17} except that the challenger forms the penultimate onion layers $\vec{O}_{\ell-1}$ using the PRP keyed with $k_{\ell-1}$ instead of a truly random permutation. Note that Hybrid_{18} is indistinguishable to the case when $b = 1$.

The hybrid argument for the case where the challenge onion is recoverable, and the recipient is honest is the same as above, except that, in Hybrid_{17}, only the message is $\perp$ (the recipient remains R). When the recipient is adversarial, the hybrid argument is just Hybrid_0-Hybrid_{15} above (without Hybrid_{16}-Hybrid_{18}). $\square$

5 Our Onion Routing Protocol, Π_t

5.1 Choosing the onion parameters

We describe our anonymous onion routing protocol, Π_t.

Let $\mathsf{TOES} = (\mathsf{KeyGen}, \mathsf{FormOnion}, \mathsf{PeelOnion}, \mathsf{PeelOnionHelper}, \mathsf{BruiseOnion})$ be the bruisable onion encryption scheme in Section 4. Let ℓ_1 be the number of mixers on the routing path, let ℓ_2 be the number of gatekeepers, and let ℓ_3 be the (expected) number of onions at each intermediary per hop. Let F_1 and F_2 be pseudorandom functions (PRFs) such that F_1 outputs zero with frequency $(\ell_1+\ell_2)\ell_3/|\mathsf{Parties}| = (\ell_1+\ell_2)\ell_3/N = \Omega(\mathsf{polylog}\,\lambda)/N$, and the range of F_2 is superpolynomial in the security parameter λ. We assume a setup with a public key infrastructure (PKI); note that the PKI enables each pair of parties (P_i, P_k) to set up a shared secret key $\mathsf{sk}_{i,k}$,e.g., by using Diffie-Hellman key exchange.

For each sender P_i, let σ_i denote the input for P_i. For each $(m_i, R_i) \in \sigma_i$, party P_i forms an onion bearing the message m_i to the recipient R_i. Additionally, P_i forms a polylog (in the security parameter) number of checkpoint onions.

The algorithms for forming the onions are essentially those of Π_a [ALU18], except we use tulip bulbs instead of standard ones. Specifically, we use tulip bulbs with $\ell_1 = \Omega(\mathsf{polylog}\,\lambda)$ mixers per onion, $\ell_2 = \Omega(\mathsf{polylog}\,\lambda)$ gatekeepers per onion, and $d = \theta\ell_1$ key-blocks per sepal. For completeness, we describe these algorithms in detail below.

Forming the message-bearing onions. To form the message-bearing onion for the message-recipient pair $(m_i, R_i) \in \sigma_i$, P_i first samples $\ell_1 + \ell_2$ parties $M_1, \ldots, M_{\ell_1}$, $G_1, \ldots, G_{\ell_2}$ uniformly at random and then runs the onion-forming algorithm $\mathsf{FormOnion}$ on the message m_i, the routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R_i)$, the public keys associated with the parties in $\vec{P}$ (which we will denote $\mathsf{pk}(\vec{P})$), and the sequence $\vec{\bot} = (\overbrace{\bot, \bot, \ldots, \bot}^{\ell_1+\ell_2 \text{ times}})$ of metadata. Here, "$\bot$" denotes the empty metadata. See Figure 2 below for the pseudocode.

1 :	$M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2} \leftarrow\!\!\$\ \mathsf{Parties}$
2 :	$\vec{\bot} = (\overbrace{\bot, \bot, \ldots, \bot}^{\ell_1+\ell_2 \text{ times}})$
3 :	$\vec{O} \leftarrow \mathsf{FormOnion}(m, (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R_i), \mathsf{pk}(\vec{P}), \vec{\bot})$

Figure 2: Pseudocode for forming the message-bearing onion

Forming the checkpoint onions. Next, P_i forms the checkpoint onions. P_i initializes the sets of nonces, $\mathcal{Y}_1, \ldots, \mathcal{Y}_{\ell_1}$, to the empty set.

Then, for every pair (j, P_k) where $j \in [\ell_1]$ is a hop-index and $P_k \in \mathsf{Parties}$ is a party, P_i determines whether or not they should form an onion for party P_k to be verified in the j^{th} hop. This is done by computing the pseudorandom function F_1 on the shared key $\mathsf{sk}_{i,k}$ and the hop-index j. If the output equals zero, P_i sets the checkpoint nonce y to $F_2(\mathsf{sk}_{i,k}, j)$; adds y to the nonce-set $\mathcal{Y}_j$; samples $\ell_1 + \ell_2 + 1$ parties $M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R$ uniformly at random; and forms a checkpoint onion by running $\mathsf{FormOnion}$ on the empty message "$\bot$," the routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$, the public keys $\mathsf{pk}(\vec{P})$ associated with the parties on the path, and the sequence $\vec{y} = (\overbrace{\bot, \bot, \ldots, \bot}^{j-1 \text{ times}}, y, \overbrace{\bot, \bot, \ldots, \bot}^{\ell_1+\ell_2-j \text{ times}})$ of metadata. See Figure 3 below for the pseudocode.

5.2 Routing onions

After forming the onions, P_i releases them into the network. From this point on, P_i acts as an intermediary or recipient. That is, P_i first sends each of its onions to the first party on the onion's routing path and then waits to receive onions. In contrast to the setup for Π_a, here, the honest parties must determine when to send out batch-processed onions without relying on a global clock; accordingly, our protocol for processing and routing tulip bulbs (i.e., onions) differs from that of Π_a.

To begin with, P_i sets counters $c_1, \ldots, c_{\ell_1}, j$ to zero.

$\mathcal{Y}_1, \ldots, \mathcal{Y}_{\ell_1} \leftarrow \emptyset$
for $(j, P_k) \in [\ell_1] \times \mathsf{Parties}$:
  if $F_1(\mathsf{sk}_{i,k}, j) = 0$:
    $y \leftarrow F_2(\mathsf{sk}_{i,k}, j)$
    $\mathcal{Y}_j \leftarrow \mathcal{Y}_j.\mathsf{append}(y)$
    $\vec{y} = (\overbrace{\perp, \perp, \ldots, \perp}^{j-1 \text{ times}}, y, \overbrace{\perp, \perp, \ldots, \perp}^{\ell_1+\ell_2-j \text{ times}})$
    $M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R \leftarrow\!\$\ \mathsf{Parties}$
    $\vec{O} \leftarrow \mathsf{FormOnion}(\perp, (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R), \mathsf{pk}(\vec{P}), \vec{y})$

Figure 3: Pseudocode for forming checkpoint onions

Upon receiving an onion O, P_i processes it: That is, P_i first peels the onion. P_i drops the onion if this produces a layer key that P_i has seen before; that is, the layer key also serves as a session id for preventing replay attacks. What happens next depends on P_i's role:

- (**Role** = Recipient) If the peeled onion O' is a message for P_i, P_i outputs it.
- (**Role** = Gatekeeper) If P_i is a gatekeeper for O and peeling O produces a peeled onion O' and a destination P' for O', P_i sends O' to P' right away. (Note that if P_i is the last gatekeeper on the routing path, P_i can recover the identity of the recipient R and the onion for R only if a sufficiently small number of mixers bruised the onion en route. See Section 4 to recall how the onion encryption construction works and its security properties.)
- (**Role** = Mixer) Otherwise if P_i is a mixer for O, P_i determines whether O was received "on time" or not (relative to P_i's internal clock). If O arrived late, P_i bruises the onion O and immediately sends the bruised onion O'' to its next destination. If P_i is the last mixer on the routing path (i.e., $P_i = M_{\ell_1}$), P_i sends the peeled onion O' to the first gatekeeper G_1.

  Otherwise if O is either early or on time, P_i places the peeled onion O' (along with its next destination P') in its message outbox. If processing O reveals the non-empty nonce $y \neq \perp$, then P_i first checks whether y belongs in a set $\mathcal{Y}_k$. (Recall from Section 5.1 that $\mathcal{Y}_k$ is the set of k^{th} layer checkpoint nonces P_i expects to see from the onions it receives.) If it does, then P_i increments c_k by one, and updates $\mathcal{Y}_k$ to exclude y.

  Upon processing sufficiently many j^{th} layer onions (i.e., if $c_j \geq \tau|\mathcal{Y}_j|$ where $0 < \tau \leq 1$ is a system parameter), P_i sends out these onions (but not the onions for future hops) in random order, and advances its local clock (i.e., increments j by one). Note that onions are shuffled at honest intermediaries when they are batch-processed and sent out in random order. See Figure 4 for the pseudocode.

6 Provable Guarantees

Recall the system parameters set forth in the Preliminaries section: χ is the constant corruption rate. That is, we assume that the adversary can corrupt up to a χ fraction of the parties. γ is the constant drop rate. An onion is *indistinguishable* if it was formed by an honest party and either bears a message or is a checkpoint onion for verification by an honest party; for our result on guaranteed message delivery, we assume that the adversary can drop up to γ fraction of indistinguishable onions. (Note that onions for verification by adversarial parties are distinguishable from other onions when the adversary observes the checkpoint values.)

```
c_1, ..., c_{ℓ_1}, j ← 0
upon receiving O
  (Role, k, y, O', P') ← PeelOnion(sk(P_i), O)
  if Role = Recipient
    return O'
  if k < j
    if Role = Gatekeeper
      send O' to P'
    else    // Role = Mixer
      O'' ← BruiseOnion(sk(P_i), O)
      send O'' to P'
  else    // k ≥ j
    place (O', P') in outbox
    if (y ≠ ⊥) ∧ (∃k s.t. y ∈ 𝒴_k)
      𝒴_k ← 𝒴_k \ {y}
      c_k ← c_k + 1
upon c_j ≥ τ|𝒴_j|
  j ← j + 1
  send peeled j^th layer onions out in random order
```

Figure 4: Pseudocode for processing onions

Recall the onion encryption parameters, ℓ_1, ℓ_2, θ, and the onion routing parameter, ℓ_3, τ, from Sections 4-5: ℓ_1 is the number of mixers on a routing path. ℓ_2 is the number of gatekeepers on a routing path. θ is the upper bound on the fraction of onion layers that can be bruised before the innermost onion becomes unrecoverable. ℓ_3 is the expected number of onions processed at an intermediary and hop. τ is the fraction of checkpoints needed to progress the local clock to the next hop. See Table 1 for a quick reference to the variables.

		Description
χ		Fraction of nodes that $\mathcal{A}$ can corrupt
γ		Fraction of (indistinguishable) onions that $\mathcal{A}$ can drop
ℓ_1	$= \Omega(\mathsf{polylog}\,\lambda)$	Number of planned mixers on a routing path
ℓ_2	$= \Omega(\mathsf{polylog}\,\lambda)$	Number of planned gatekeepers on a routing path
θ	$> \frac{1}{2} + \chi$	Fraction of layers in an onion that cannot be bruised
ℓ_3	$= \Omega(\mathsf{polylog}\,\lambda)$	Expected number of onions per intermediary per hop
τ	$< (1-\gamma)(1-\chi)$	Fraction of checkpoints needed to progress

Table 1: Table of adversary and system parameters.

We present the provable guarantees for our protocol, Π_t. We show that when we set the parameters as in Table 1, Π_t delivers at least (arbitrarily close to) $\left(\frac{1/2+\tau-2\theta}{1-\theta}\right)\left(1-\mathcal{O}\left(\frac{1}{\mathsf{polylog}\,\lambda}\right)\right)-\gamma$ fraction of the honest parties' messages differentially privately. For small constants χ, γ (e.g., 10% corruption rate and 10% drop rate), this translates to a constant fraction message delivery rate. In a more reasonable setting where at most 5% of the parties are adversarial and maliciously bruising onions, and with 0% drop rate, Π_t guarantees a much higher message delivery rate of over 0.85; and as the corruption rate goes to 0, the message delivery rate tends to 1. One cannot expect much better solutions since, even in the synchronous setting, the adversary can always bring down the message delivery rate by dropping sufficiently many onions (from known lower bounds [DMMK18], the round complexity of anonymous protocols is at least polylogarithmic in the security parameter, which implies that every randomly chosen routing path includes a corrupted party with overwhelming probability).

In the proofs, we make ample use of the following fact, which is a corollary of the Azuma-Hoeffding inequality [MU05, Theorem 13.7]: Let B be a set of marbles. Let S be a random sample with or without replacement of the marbles, and let X be the number of red marbles in the sample S. If the expected number of red marbles in the sample, $\mathbb{E}[X]$, is at least polylog in the security parameter, then with probability $1 - e^{-\Omega(\mathsf{poly}(\lambda))}$, $X \in \mathbb{E}[X](1 \pm \mathcal{O}((\mathsf{polylog}(\lambda))^{-1})$. For brevity, we write that a random variable X is w.o.p. arbitrarily close to a value V if $\Pr\left[X \notin V(1 \pm \mathcal{O}((\mathsf{polylog}\,\lambda)^{-1}))\right] = e^{-\Omega(\mathsf{poly}(\lambda))}$.

6.1 Proof of message delivery rate

We first prove that Π_t guarantees a constant fraction message delivery rate in the regime where $(1 + 2\tau - 4\theta)\left(1 - \mathcal{O}\left(\frac{1}{\mathsf{polylog}\,\lambda}\right)\right) > 2\gamma(1 - \theta)$. Specifically,

Theorem 2. *A run of protocol Π_t with parameters $\ell_1 = \Omega(\mathsf{polylog}\,\lambda)$, $\ell_2 = \Omega(\mathsf{polylog}\,\lambda)$, $\ell_3 = \Omega(\mathsf{polylog}\,\lambda)$, $\theta > \frac{1}{2} + \chi$, $\tau < 1 - \gamma(1 - \chi) - \chi$, and $(1 + 2\tau - 4\theta)\left(1 - \mathcal{O}\left(\frac{1}{\mathsf{polylog}\,\lambda}\right)\right) > 2\gamma(1 - \theta)$, delivers at least*

$$\left(\frac{1/2 + \tau - 2\theta}{1 - \theta}\right)\left(1 - \mathcal{O}\left(\frac{1}{\mathsf{polylog}\,\lambda}\right)\right) - \gamma > 0$$

fraction of the honest parties' messages with overwhelming probability.

Proof. Let $j \in [\ell_1]$ be a hop-index, and P_k a party. Let $\mathcal{C}_{j,k}$ be the set of checkpoint values that P_k expects to observe during hop j. Since the number of parties is $\mathcal{O}(\mathsf{poly}\,\lambda)$, $\ell_1, \ell_2 \in \Omega(\mathsf{polylog}\,\lambda)$, and intermediate parties on onions' routes are chosen uniformly at random, w.o.p. for all j and k, the actual number of checkpoint values with P_k at hop j is arbitrarily close to its expectation, $\mathbb{E}[|\mathcal{C}_{j,k}|]$. Thus, in the remainder of the proof, w.l.o.g., we can use the expectations of all these values.

We first need to show that under the conditions of the theorem, the protocol at each party progresses through all the hops of the protocol. Indeed, for every hop-index $j \in [\ell_1]$ and honest party P_k, w.o.p., the adversary can drop up to approximately γ fraction of the indistinguishable checkpoints in $\mathcal{C}_{j,k}$ (Azuma-Hoeffding inequality), plus all of the other checkpoints (the non-indistinguishable ones that the adversarial parties are supposed to send to P_k). Thus, w.o.p., P_k is guaranteed to eventually receive sufficiently many onions in $\mathcal{C}_{j,k}$ to progress to the next hop (i.e., P_k receives at least slightly less than the expected $1 - \gamma(1 - \chi) - \chi = (1 - \gamma)(1 - \chi)$ fraction of the onions in $\mathcal{C}_{j,k}$).

An onion doesn't make it to its final destination for one of two reasons: either the onion was dropped by the adversary (reason 1), or it was too bruised to be reconstructed at the penultimate step (reason 2). The adversary can maximize the total number of onions that don't make it by not overlapping onions that don't make it because of reason 1 and those that don't because of reason 2. That is, the adversary doesn't waste a bruising on an onion that they will ultimately drop.

The fraction of onions dropped by the adversary is bounded by γ. Next we compute the fraction of onion that arrived too bruised at the penultimate step. To bound this number we first bound the total number of bruises of all onions in all iterations of the protocol.

Let us first bound the fraction of the j^{th} layers of indistinguishable onions that P_k bruises. If P_k is honest, they will follow the protocol and only bruise (the j^{th} layers of) onions they receive after observing τ fraction of the values in $\mathcal{C}_{j,k}$. The adversary can fix the schedule so that P_k receives checkpoints in $\mathcal{C}_{j,k}$ from the adversarial parties first. Even so, w.o.p., the number of checkpoint values in onions formed by adversarial parties, $A_{j,k}$, is arbitrarily close to the expected number $\mathbb{E}[A_{j,k}]$ (Azuma-Hoeffding inequality). Likewise, w.o.p., the number of checkpoint values in indistinguishable onions, $H_{j,k}$, is arbitrarily close to the expected number $\mathbb{E}[H_{j,k}]$ (Azuma-Hoeffding inequality). It follows that w.o.p., P_k observes at least (arbitrarily close to) $(\tau-\chi)|\mathcal{C}_{j,k}| = (\tau-\chi)(A_{j,k}+H_{j,k})$ checkpoints values embedded in indistinguishable onions. This translates to P_k observing at least (arbitrarily close to) $\frac{\tau-\chi}{1-\chi}$ of the checkpoints values embedded in indistinguishable onions "on time."

In contrast, an adversarial party can bruise every onion layer it processes.

Thus, the total fraction of *layers* of indistinguishable onions that will be bruised is bounded above by the expression: (fraction bruised when in honest parties) $\times$ (fraction of honest parties) + (fraction bruised while in corrupted party) $\times$ (fraction corrupted parties) i.e., w.o.p., at most (arbitrarily close to)

$$\begin{aligned}\left(1-\frac{\tau-\chi}{1-\chi}\right)(1-\chi)+1\cdot\chi &= \left(\frac{1-\chi-\tau+\chi}{1-\chi}\right)(1-\chi)+\chi \\ &= 1-\tau+\chi \qquad (1)\end{aligned}$$

An onion is too bruised (i.e., the innermost layer of the onion cannot be recovered) if it is bruised too many times (i.e., for $>\theta$ fraction of the bruisable layers). Thus, from (1), the adversary can sufficiently bruise, w.o.p., at most arbitrarily close to $(1-\tau+\chi)/(1-\theta) \le (1/2-\tau+\theta)/(1-\theta)$ fraction of the indistinguishable *onions*.

This leaves at least arbitrarily close to $1-\left(\frac{1/2-\tau+\theta}{1-\theta}+\gamma\right) = \left(\frac{1/2+\tau-2\theta}{1-\theta}\right)-\gamma$ fraction of message-bearing onions being both "originating from honest parties" and "ultimately delivered" (Azuma-Hoeffding inequality). □

Remark on censorship. An adversary can censor a party in our protocol by delaying just that party's onions and causing them to be too bruised and eventually undelivered. This is the only way to achieve anonymity: if these delayed onions were ever delivered, they would be de-anonymized. Thus, the lack of censorship resilience is inherent to the asynchronous model. Moreover, note that in the asynchronous model where the adversary controls all the links, censorship is always within the adversary's power (even in a protocol that eventually delivers all messages) since the messages that the adversary aims to censor can be delayed until other parts of the computation are done; so even if they are eventually delivered, the adversary can make sure that by the time they arrive, they are no longer useful for whatever protocol the honest participants need them for. Giving the adversary in our protocol the ability to cause them to be dropped altogether does not provide the adversary extra power.

Here, we prove that Π_t is computationally differentially private.

Theorem 3. *For any constant $\epsilon > 0$, Π_t with parameters $\ell_1 = \Omega(\mathsf{polylog}\,\lambda)$, $\ell_2 = \Omega(\mathsf{polylog}\,\lambda)$, $\ell_3 = \Omega(\mathsf{polylog}\,\lambda)$, and $\theta > \frac{1}{2}+\chi$ is computationally $(\epsilon, \mathsf{negl}(\lambda))$-differentially private from the adversary who corrupts up to $\chi < \frac{1}{2}$ fraction of the parties and drops any fraction $0 \le \gamma \le 1$ of the indistinguishable onions.*

Proof. We prove below that Π_t achieves (statistical) $(\epsilon, \mathsf{negl}(\lambda))$-differential privacy for any constant $\epsilon > 0$ when the PRFs F_1 and F_2 are truly random functions, and the underlying bruisable onion scheme is perfectly secure.[5] From Canetti's UC composition theorem [Can01], this implies that Π_t is computationally differentially private when we use PRFs and our onion encryption scheme from Section 4 instead.

Let σ_0, σ_1 be any neighboring input vectors. That is, σ_0 and σ_1 are identical except on the inputs of two honest senders P_i and P_j and the "outputs" of two receivers P_u and P_v. On input vector σ_0, P_i sends a message to P_u, and honest P_j sends a message to P_v; while in σ_1, this is swapped (P_i sends to P_v, while P_j sends to P_v). For $b \in \{0,1\}$, let $(I_{i,1}, \dots, I_{i,\ell_1+\ell_2}, R_{b,i})$ be the routing path that P_i picks for their message-bearing onion, and let $(I_{j,1}, \dots, I_{j,\ell_1+\ell_2}, R_{b,j})$ be the routing path that P_j picks for their message-bearing onion.

We prove the theorem by cases.

Case 1: neither P_i's message nor P_j's message is delivered. The only difference between the scenario when the input vector is σ_0 and the scenario when it is σ_1 is the receivers for P_i and P_j's challenge messages. Everything else is identically distributed. Thus, in this case, the adversarial views for the two settings are perfectly indistinguishable since the adversary never observes the challenge onions' layers for P_u and P_v, i.e., $\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_0) = \mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_1)$.

Case 2: both P_i's message and P_j's message is delivered. In this case, $\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_0)$ and $\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_1)$ are statistically indistinguishable, i.e., the total variation distance between $\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_0)$ and $\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_1)$ is negligible in the security parameter, from Lemma 1 below (proven in the next subsection):

Lemma 1. *Let $O = (O_1, \dots, O_{\ell_1+\ell_2+1})$ and $O' = (O'_1, \dots, O'_{\ell_1+\ell_2+1})$ be any two message-bearing onions that were formed by honest parties that make it to their final destinations. Let P be the origin (the honest sender) of O, and let P' be the origin of O'. Let $i_1 < \cdots < i_w \le \ell_1$ be the hop-indices where O shuffles with other onions (i.e., arrives on time or early at an honest party), and let $i'_1 < \cdots < i'_{w'} \le \ell_1$ be the moments when O shuffles with other onions. (1) W.o.p., there exists a positive constant $c > 0$ such that $|\mathcal{I}| = |\{i_1, \dots, i_w\} \cap \{i'_1, \dots, i'_{w'}\}| \ge c\ell_1$. (2) Let $r = \max \mathcal{I} = \max\{i_1, \dots, i_w\} \cap \{i'_1, \dots, i'_{w'}\}$ be the last time that both O and O' shuffle. Given the unordered set $\{O_r, O'_r\}$, the adversary can correctly match P to O_r and P' to O'_r with probability only negligibly greater than $1/2$.*

Case 3: P_i's message or P_j's message is delivered. In this case, $\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_0)$ and $\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_1)$ are differentially private; in other words, $\Pr\big[\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_0) \in \mathcal{V}\big] \le e^{\epsilon} \Pr\big[\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_1) \in \mathcal{V}\big] + \mathsf{negl}(\lambda)$ for every set $\mathcal{V}$ of views. W.l.o.g., we assume that P_i's message makes it to its recipient $R_{b,i}$, but $R_{b,j}$ does not receive P_j's message. Let r be the final hop at which O shuffles with other onions. The indistinguishable onions, including the message-bearing onion O from P_i to $R_{b,i}$, are sufficiently shuffled together by hop r by Lemma 2 below:

Lemma 2. *Let $O = (O_1, \dots, O_{\ell_1+\ell_2+1})$ be any indistinguishable onion. If O shuffles with other onions a polylog (in the security parameter) number of times before some hop r, then given O_r and any r^{th} layer indistinguishable onion O'_r in the adversarial view, the adversary can correctly guess which is the evolved version of O_1 with probability only negligibly greater than one-half.*

Since the adversary cannot determine the origin of any indistinguishable onion at hop r (from the above claim), the only information the adversary has to help determine the input setting is the volumes of onions received by each recipient. W.o.p., the number n of indistinguishable

[5] That is, we assume that the adversary cannot determine any meaningful information "hidden behind an honest party," e.g., the adversary cannot determine the message or the rest of the routing path of an onion that goes into an honest intermediary; see Section 3.3.1 for more details. Further, we assume that the gatekeepers always drop an onion with too many bruises ($> \theta\ell_1$) since w.o.p., at least one of the $\ell_2 = \Theta(\mathsf{polylog}\,\lambda)$ gatekeepers in each onion is honest.

checkpoint onions for either P_u or P_v is arbitrarily close to the expected number $\mathbb{E}[n]$ since $\mathbb{E}[n]$ is polylogarithmic in the security parameter (Azuma-Hoeffding inequality). Seen this way, the number of indistinguishable checkpoint onions for P_u, which we denote by n_u, and the number of indistinguishable checkpoint onions for P_v, which we denote by n_v, are Binomial random variables with n trials and bias $\frac{1}{2}$, i.e., $n_u, n_v \leftarrow \mathsf{Binomial}(n, \frac{1}{2})$. Thus, the numbers of messages received are obscured by a Binomial Mechanism which, for $n = \Omega(\mathsf{polylog}\,\lambda)$, was shown [DKM+06] to be $(\epsilon/2, \mathsf{negl}(\lambda))$-differentially private for any positive constant ϵ. It follows from the composition theorem for differential privacy that Π_t achieves (computational) $(\epsilon, \mathsf{negl}(\lambda))$-differential privacy for any positive constant ϵ. □

Recall neighboring input vectors: σ_0 and σ_1 are neighboring if they are the same except for a pair of messages to be sent from honest senders and received by honest recipients. We note that, from the composition theorem for differential privacy, Theorem 3 holds even if we loosen this notion. Specifically,

Corollary 1. *Let the swap-distance $d(\sigma_0, \sigma_1)$ between σ_0 and σ_1 be the length (minus one) of the shortest sequence of input vectors $(\sigma_0, \sigma_{0\to1,1}, \ldots, \sigma_{0\to1,d} = \sigma_1)$. Consider Π_t with parameters $\ell_1 = \Omega(\mathsf{polylog}\,\lambda)$, $\ell_2 = \Omega(\mathsf{polylog}\,\lambda)$, $\ell_3 = \Omega(\mathsf{polylog}\,\lambda)$, and $\theta > \frac{1}{2} + \chi$. For any constant swap-distance $d \geq 0$, any small constant $\epsilon > 0$, any (computationally-bounded) adversary $\mathcal{A}$ who corrupts up to $\chi < \frac{1}{2}$ fraction of the parties, any pair of inputs σ_0 and σ_1 such that $d(\sigma_0, \sigma_1) \leq d$, and any set $\mathcal{V}$ of adversarial views,* $\Pr\left[\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_0) \in \mathcal{V}\right] \leq e^{\epsilon} \Pr\left[\mathsf{View}^{\Pi_t,\mathcal{A}}(\sigma_1) \in \mathcal{V}\right] + \mathsf{negl}(\lambda)$.

6.1.1 Proofs of lemmas.

We provide proofs for the lemmas supporting the proof of Theorem 3. As in the proof of Theorem 3, we will assume that Π_t uses the idealized counterparts of the cryptographic primitives used in the protocol.

of Lemma 1. By construction, the intermediaries $I_1, \ldots, I_{\ell_1}$ are randomly sampled from the set of all participants. Thus if the recipient for O receives the message embedded in the onion, then this implies that the message (in encrypted form) traversed the path, $\vec{I} = (I_1, \ldots, I_{\ell_1})$, that consists of many honest parties. More precisely, with overwhelming probability, the fraction of honest parties in $\vec{I}$ is a constant value arbitrarily close to $1 - \chi$ where χ denotes the corruption rate (Azuma-Hoeffding inequality).

From the threshold security property of the onion encryption scheme, the recipient can receive $O_{\ell_1+\ell_2+1}$ only if strictly more than $\frac{1}{2} + \chi$ fraction of the parties in $\vec{I}$ passed the onion on without bruising it. Since at most (an arbitrarily close to) χ fraction of the parties in $\vec{I}$ are adversarial, the fraction of the parties in $\vec{I}$ that are honest and sent O shuffled among other onions is strictly greater than one-half. With overwhelming probability, each time the onion O mixes, it does so with a number of onions that is arbitrarily close to the expected polylogarithmic value (Azuma-Hoeffding inequality).

Following a similar argument, O' also mixed with a polylog number of onions, some constant fraction $> \frac{1}{2}$ of the times (in $\vec{I'}$). Thus, by the pigeonhole principle, both O and O' were mixed together with a polylog number of onions a polylog number of times before the r^{th} hop. From Lemma 2 (above with proof below), it is possible to show that by the r^{th} hop, the adversary "loses track" of which onion is which: given the unordered set $\{O_r, O'_r\}$, the adversary can correctly match P to O_r and P' to O'_r with probability only negligibly greater than that of a random guess. □

of Lemma 2. This is a generalization of the proof of mixing in the synchronous setting without any adversarial parties [ALU18, Theorem 10]. We prove that by the r^{th} hop, the adversary "loses track of where O is;" that is, given O_r and any r^{th} layer indistinguishable onion O'_r, the adversary can correctly guess which is the evolved version of O_1 with probability only negligibly greater than one-half. We show this to be true even when the challenger reveals some of the times that O mixes before the r^{th} hop. Specifically, the challenger reveals all the times $i_{s,1} < i_{e,1} < i_{s,2} < i_{e,2} < \cdots < i_{s,L} < i_{s_L} \leq r$ such that each "cycle" $(i_{s,j}, \ldots, i_{e,j})$ starts and ends with "good hops" (hops in which O mix with other onions) and having some constant number of "bad hops" (hops in which O doesn't mix in between these hops).

For any cycle $(i_{s,j}, \ldots, i_{e,j})$, the challenger essentially tells the adversary that O is in the set $\mathcal{O}_{s,j}$ of onions that mix at hop $i_{s,j}$ and then doesn't mix again until hop $i_{e,j}$. The adversary has some idea of which onion in $\mathcal{O}_{s,j}$ is O, represented by a probability distribution over the space $\mathcal{O}_{s,j}$. (Note that some of these probabilities may be zero.) Let $\mathcal{O}_{s,j,1}, \mathcal{O}_{s,j,2}, \mathcal{O}_{s,j,3}, |\mathcal{O}_{s,j,1}| = |\mathcal{O}_{s,j,2}| = |\mathcal{O}_{s,j,3}| = \frac{1}{3}|\mathcal{O}_{s,j}|$ be a partition of the onions in $\mathcal{O}_{s,j}$ such that $\mathcal{O}_{s,j,1}$ is the set of onions in $\mathcal{O}_{s,j}$ that are most likely to be O (according to the adversary's belief), $\mathcal{O}_{s,j,3}$ is the set of onions in $\mathcal{O}_{s,j}$ that are the least likely to be O, and $\mathcal{O}_{s,j,2}$ is the set of all other onions in $\mathcal{O}_{s,j}$. For $k \in [3]$, let $Z_{s,j,k}$ and $z_{s,j,k}$ be, respectively, the probability of the most likely onion in $\mathcal{O}_{s,j,k}$ and the probability of the least likely onion in $\mathcal{O}_{s,j,k}$; so $Z_{s,j,1} \geq z_{s,j,1} \geq Z_{s,j,2} \geq z_{s,j,2} \geq Z_{s,j,3} \geq z_{s,j,3}$.

The adversary can corrupt fewer than a constant fraction of the parties, and we will assume w.l.o.g. that the adversary corrupts as many parties that they can. Thus, it follows that each party (at their own local) time $i_{e,j}$ receives arbitrarily close to the expected number of onions from each of the sets $\mathcal{O}_{s,j,1}, \mathcal{O}_{s,j,2}, \mathcal{O}_{s,j,3}$ (Lemma 3; below). Thus, the probabilities $Z_{e,j,1}, z_{e,j,3}$ of the most likely and least likely onions in $\mathcal{O}_{e,j}$ (coming out mixed at hop $i_{e,j}$) is bounded as follows: for any constant $\epsilon > 0$, $Z_{e,j,1} \leq \frac{1+\epsilon}{3} \sum_{k=1}^{3} Z_{s,j,k}$ and $z_{e,j,1} \geq \frac{1-\epsilon}{3} \sum_{k=1}^{3} z_{s,j,k}$. It follows that the "gap" $G_{e,j} = Z_{e,j,1} - z_{e,j,3}$ between these probabilities is at most half of the gap $G_{s,j} = Z_{s,j,1} - z_{s,j,3}$ between the most likely and least likely probabilities for the prior cycle. From the pigeonhole principle, the number of cycles, k, is polylog in the security parameter. This proves that the difference in probabilities becomes negligible by hop $i_{e,L} \leq r$. □

Lemma 3. *(In the proof of Lemma 2) w.o.p., each honest party at hop $i_{e,j}$ receives arbitrarily close to the expected number of onions from each of the sets $\mathcal{O}_{s,j,1}$, $\mathcal{O}_{s,j,2}$, and $\mathcal{O}_{s,j,3}$.*

of Lemma 3. Let $\mathcal{O}_{\mathsf{all}}$ be the set of all onions that mix at hop $i_{s,j}$. The onion $O \in \mathcal{O}_{\mathsf{all}}$ mixes with a polylog (in the security parameter) onions at one of the honest parties. From Azuma-Hoeffding inequality, w.o.p., $|\mathcal{O}_{\mathsf{all}}| = \mathcal{O}(N \,\mathsf{polylog}\, \lambda)$ where N denotes the the number of parties, and λ denotes the security parameter.

Recall from the proof of Lemma 2 that $\mathcal{O}_{s,j}$ is the set of onions in $\mathcal{O}_{\mathsf{all}}$ that never mix between hops $i_{s,j}$ and $i_{e,j}$, exclusively, because they always either route through corrupted parties or arrive too late at honest ones. Since both the corruption rate and the number of hops between $i_{s,j}$ and $i_{e,j}$ are at most constant terms, it follows that $|\mathcal{O}_{s,j,1}| = |\mathcal{O}_{s,j,2}| = |\mathcal{O}_{s,j,3}| = \frac{1}{3}|\mathcal{O}_{s,j}| = \mathcal{O}(N \,\mathsf{polylog}\, \lambda)$.

Let P be any honest party at hop $i_{e,j}$. For each $1 \leq k \leq 3$, each onion in $\mathcal{O}_{s,j,k}$ routes to P with fixed probability $\frac{1}{N}$ regardless of where the other onions go. Thus, the number U_k of onions that route from $\mathcal{O}_{s,j,k}$ to P can be expressed as a binomial random variable with expectation $\mathbb{E} = \frac{|\mathcal{O}_{s,j}|}{3N} = \mathsf{polylog}\, \lambda$. Again using the Azuma-Hoeffding inequality, it follows that U_k is arbitrarily close to its expected value $\mathbb{E}$ w.o.p. □

7 Conclusion and Open Problems

We present the first provably anonymous communication protocol in an asynchronous environment. Our protocol guarantees differential privacy of the sources and destinations information of the messages under a strong adversity model. The adversary fully controls the schedule of delivery of all messages, can corrupt a constant fraction of the parties, and drop a constant fraction of all messages.

While our work proves the possibility of anonymity in a fully asynchronous network, many further question were left open for further research. In particular we are interested in stronger privacy models than just differential privacy, and in anonymous bidirectional communication in a dynamic network with node churn.

Our work also raised interesting questions regarding the inherent vulnerability of asynchronized communication to adversarial attacks and inherent gaps in security between synchronized and asynchronized models.

References

[ACLM22] M. Ando, M. Christ, A. Lysyanskaya, and T. Malkin. Poly onions: Achieving anonymity in the presence of churn. In *TCC 2022, Part II*, *LNCS* 13748. Springer, Heidelberg, November 2022.

[AL21] M. Ando and A. Lysyanskaya. Cryptographic shallots: A formal treatment of repliable onion encryption. In *TCC 2021, Part III*, *LNCS* 13044. Springer, Heidelberg, November 2021.

[ALU18] M. Ando, A. Lysyanskaya, and E. Upfal. Practical and provably secure onion routing. In *ICALP 2018*, *LIPIcs* 107. Schloss Dagstuhl, July 2018.

[ALU21] M. Ando, A. Lysyanskaya, and E. 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.

[AMWB23] R. Attarian, E. Mohammadi, T. Wang, and E. H. Beni. Mixflow: Assessing mixnets anonymity with contrastive architectures and semantic network information. *Cryptology ePrint Archive*, 2023.

[BFT04] R. Berman, A. Fiat, and A. Ta-Shma. Provable unlinkability against traffic analysis. In *FC 2004*, *LNCS* 3110. Springer, Heidelberg, February 2004.

[BKM+13] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi. Anoa: A framework for analyzing anonymous communication protocols. In *Computer Security Foundations Symposium (CSF), 2013 IEEE 26th*. IEEE, 2013.

[Bra84] G. Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, 1984.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*. IEEE Computer Society Press, October 2001.

[CBM15] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015.

[Cha81] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cha88] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, January 1988.

[CL05] J. Camenisch and A. Lysyanskaya. A formal treatment of onion routing. In *CRYPTO 2005*, *LNCS* 3621. Springer, Heidelberg, August 2005.

[CR93] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th ACM STOC*. ACM Press, May 1993.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO'98*, *LNCS* 1462. Springer, Heidelberg, August 1998.

[DKM+06] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT 2006*, *LNCS* 4004. Springer, Heidelberg, May / June 2006.

[DMMK18] D. Das, S. Meiser, E. Mohammadi, and A. Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018.

[DMNS06] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC 2006*, *LNCS* 3876. Springer, Heidelberg, March 2006.

[DMS04] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, 2004.

[IKK05] J. Iwanik, M. Klonowski, and M. Kutyłowski. Duo-onions and hydra-onions—failure and adversary resistant onion protocols. In *Communications and Multimedia Security*. Springer, 2005.

[KBS+19] C. Kuhn, M. Beck, S. Schiffner, E. A. Jorswieck, and T. Strufe. On privacy notions in anonymous communication. *Proc. Priv. Enhancing Technol.*, 2019(2):105–125, 2019.

[KBS20] C. Kuhn, M. Beck, and T. Strufe. Breaking and (partially) fixing provably secure onion routing. In *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020.

[KCDF17] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 2017.

[KHRS21] C. Kuhn, D. Hofheinz, A. Rupp, and T. Strufe. Onion routing with replies. In *ASIACRYPT 2021, Part II*, *LNCS* 13091. Springer, Heidelberg, December 2021.

[Lyn96] N. A. Lynch. *Distributed algorithms*. Elsevier, 1996.

[MD05] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2005.

[MPRV09] I. Mironov, O. Pandey, O. Reingold, and S. P. Vadhan. Computational differential privacy. In *CRYPTO 2009*, *LNCS* 5677. Springer, Heidelberg, August 2009.

[MU05] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[PHE+17] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The loopix anonymity system. In *26th usenix security symposium (usenix security 17)*, 2017.

[Rop21] L. Ropek. Someone is running hundreds of malicious servers on the Tor network and might be de-anonymizing users. `https://tinyurl.com/2p999e8e`, December 2021.

[RS93] C. Rackoff and D. R. Simon. Cryptographic defense against traffic analysis. In *25th ACM STOC*. ACM Press, May 1993.

[SEV+15] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. RAPTOR: Routing Attacks on Privacy in Tor. In *USENIX Security Symposium*, 2015.

[TGL+17] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 2017.

[vdHLZZ15] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*. ACM, 2015.

[WSJ+18] R. Wails, Y. Sun, A. Johnson, M. Chiang, and P. Mittal. Tempest: Temporal dynamics in anonymity systems. *PoPETs*, 2018(3):22–42, 2018.

A Full proof of Theorem 1

A.1 Reductions for $\mathsf{Hybrid}_0, \ldots, \mathsf{Hybrid}_7$

$\mathsf{Hybrid}_0, \ldots, \mathsf{Hybrid}_7$ convert the "outermost layers" of the challenge onion in Experiment 0 to those in Experiment 1.

Lemma 4. Hybrid_0 *and* Hybrid_1 *are indistinguishable.*

Proof. Assume that there exists a p.p.t. adversary $\mathcal{A}$ that can distinguish whether it is in Hybrid_0 or Hybrid_1 with non-negligible (in the security parameter) advantage. We construct the following reduction $\mathcal{B}$ that breaks the CCA2 security of the underlying cryptosystem in that case. $\mathcal{B}$ plays the challenger for $\mathcal{A}$ and the adversary in the CCA2 security game against its challenger $\mathcal{C}$.

1. During setup, $\mathcal{A}$ sends the names of the honest parties P_m, P_g, and P_r, along with the public keys of the adversarial parties $\mathsf{pk}(\mathsf{Bad})$ to $\mathcal{B}$.
2. $\mathcal{B}$ generates key pairs for P_g and P_r; that is, $\mathcal{B}$ obtains $(\mathsf{pk}(P_g), \mathsf{sk}(P_g))$ by running $\mathsf{KeyGen}(P_g)$ and $(\mathsf{pk}(P_r), \mathsf{sk}(P_r))$ by running $\mathsf{KeyGen}(P_r)$. $\mathcal{C}$ supplies the public key $\mathsf{pk}(P_m)$ of P_m, which $\mathcal{B}$ forwards to $\mathcal{A}$ along with the public keys $\mathsf{pk}(P_g), \mathsf{pk}(P_r)$.
3. During the first query phase, whenever $\mathcal{A}$ sends an onion $O = ((E, B), C, S)$ to be peeled (resp. bruised) on behalf of P_m, $\mathcal{B}$ sends the ciphertext portion E to $\mathcal{C}$ to be decrypted. Once $\mathcal{C}$ replies with the layer key k, $\mathcal{B}$ completes the $\mathsf{PeelOnion}$ (resp. $\mathsf{BruiseOnion}$) algorithm on O and replies to $\mathcal{A}$ with the output. (For queries to peel or bruise on behalf of P_g and P_r, $\mathcal{B}$ just runs $\mathsf{PeelOnion}$ or $\mathsf{BruiseOnion}$.)

4. During the challenge phase, $\mathcal{A}$ sends to $\mathcal{B}$ the onion parameters: the message m and the routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ such that $P_m = M_{i_1}$ and $P_g = G_{j=i_2-\ell_1}$.
5. $\mathcal{B}$ picks the layer keys $k_1, \ldots, k_\ell$ and sends the challenge messages $m_0 = k_{i_1}$ and $m_1 = 0$ to $\mathcal{C}$. Once $\mathcal{B}$ obtains the ciphertext $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}(M_{i_1}, m_b))$ from $\mathcal{C}$, it forms the challenge onion O_1 by following the $\mathsf{FormOnion}$ procedure, except in forming the ${i_1}^{th}$ onion layer, $\mathcal{B}$ uses c_b in place of the ciphertext E_{i_1}. $\mathcal{B}$ sends O_1 to $\mathcal{A}$.
6. Once again, $\mathcal{A}$ is allowed to query onions. $\mathcal{B}$ deals with these queries in the same way as before.
7. If $\mathcal{A}$ guesses Hybrid_0, $\mathcal{B}$ guesses 0. Otherwise if $\mathcal{A}$ guesses Hybrid_1, $\mathcal{B}$ guesses 1.

The reduction works because $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s, and the reduction runs in polynomial time. □

Lemma 5. Hybrid_1 *and* Hybrid_2 *are indistinguishable.*

Proof. Assume that there is a p.p.t. adversary $\mathcal{A}$ that can distinguish between being in Hybrid_1 or Hybrid_2. We construct the following reduction $\mathcal{B}$ that can win the collision resistance game with non-negligible advantage. $\mathcal{B}$ plays the challenger for $\mathcal{A}$ and the adversary in the collision resistance game against its challenger $\mathcal{C}$.

1. During setup, $\mathcal{B}$ interacts only with $\mathcal{A}$; it does not interact with $\mathcal{C}$. $\mathcal{A}$ sends the names of the honest parties P_m, P_g, P_r and the public portions $\mathsf{pk}(\mathsf{Bad})$ of the keys belonging to the adversarial parties to $\mathcal{B}$.
2. $\mathcal{B}$ generates the keys for the honest parties and sends the public portions of the generated keys to $\mathcal{A}$.
3. During the first query phase, $\mathcal{B}$ still interacts only with $\mathcal{A}$. Whenever $\mathcal{A}$ sends $\mathcal{B}$ a query to peel (resp. bruise) an onion O on behalf of an honest party P, $\mathcal{B}$ responds with the output of $\mathsf{PeelOnion}$ (resp. $\mathsf{BruiseOnion}$) on O and P's secret key.
4. During the challenge phase, $\mathcal{A}$ sends to $\mathcal{B}$ the onion parameters: the message m and the routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ such that $P_m = M_{i_1}$ and $P_g = G_{j=i_2-\ell_1}$.
5. $\mathcal{B}$ picks $k_1, \ldots, k_\ell$ and obtains E_{i_1} by encrypting the dummy key 0 under $\mathsf{pk}(M_{i_1})$. $\mathcal{B}$ then follows the $\mathsf{FormOnion}$ procedure to form O_1, except $\mathcal{B}$ uses E'_{i_1} instead of an encryption of k_{i_1} in the ${i_1}^{th}$ onion layers $\vec{O}_{i_1}$; the same way as both Hybrid_1 and Hybrid_2 do. $\mathcal{B}$ sends O_1 to $\mathcal{A}$.
6. Once again, $\mathcal{A}$ is allowed to query onion. If $\mathcal{A}$ ever produces an onion $O = ((E_{i_1}, B), C, S) \notin \vec{O}_{i_1}$ that peels to an actual onion $O' \neq \perp$, then $\mathcal{A}$ has found a collision in the hash function. $\mathcal{B}$ forwards the collision to $\mathcal{C}$ (and wins in this case).
7. Finally, $\mathcal{A}$ outputs a guess (either Hybrid_1 or Hybrid_2).

The reduction works because $\mathcal{A}$'s advantage is the probability of the event that $\mathcal{A}$ produces an an onion $O = ((E_{i_1}, B), C, S) \notin \vec{O}_{i_1}$ that peels to an actual onion $O' \neq \perp$; this is also $\mathcal{B}$'s advantage. Moreover, the reduction runs in polynomial time. □

Lemma 6. Hybrid_2 *and* Hybrid_3 *are indistinguishable.*

Proof. Assume that there exists a p.p.t. adversary $\mathcal{A}$ that can distinguish whether it is in Hybrid_2 or Hybrid_3 with non-negligible (in the security parameter) advantage. We construct the following reduction $\mathcal{B}$. $\mathcal{B}$ plays the role of the challenger for $\mathcal{A}$ and that of the adversary in the PRP security game against the challenger $\mathcal{C}$.

1. During setup, $\mathcal{B}$ interacts only with $\mathcal{A}$; it does not interact with $\mathcal{C}$. $\mathcal{A}$ sends the names of the honest parties P_m, P_g, P_r and the public portions $\mathsf{pk}(\mathsf{Bad})$ of the keys belonging to the adversarial parties to $\mathcal{B}$.

2. $\mathcal{B}$ generates the keys for the honest parties and sends the public portions of the generated keys to $\mathcal{A}$.
3. During the first query phase, $\mathcal{B}$ still interacts only with $\mathcal{A}$. Whenever $\mathcal{A}$ sends $\mathcal{B}$ a query to peel (resp. bruise) an onion O on behalf of an honest party P, $\mathcal{B}$ responds with the output of $\mathsf{PeelOnion}$ (resp. $\mathsf{BruiseOnion}$) on O and P's secret key.
4. During the challenge phase, $\mathcal{A}$ sends to $\mathcal{B}$ the onion parameters: the message m and the routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ such that $P_m = M_{i_1}$ and $P_g = G_{j=i_2-\ell_1}$.
5. $\mathcal{B}$ forms the inner onion layers $\vec{O}_\ell, \ldots, \vec{O}_{i_1+1}$ by following the $\mathsf{FormOnion}$ algorithm but deviates from the algorithm at the ${i_1}^{th}$ layers $\vec{O}_{i_1}$: First, $\mathcal{B}$ forms the ciphertext E_{i_1} by encrypting the dummy key 0 under the public key of M_{i_1}. Next, $\mathcal{B}$ sends to $\mathcal{C}$: the blocks $((M_{i_1+1}, E_{i_1+1}), B_{i_1+1,1}, \ldots, B_{i_1+1,\ell-i_1})$, the content C_{i_1+1}, and the $i_1 + 1^{st}$ layer T_{i_1+1} of the sepal for the first processor on the path. $\mathcal{C}$ replies with $(B_{i_1,1}, \ldots, B_{i_1,\ell-i_1+1}, C_{i_1}, T_{i_1})$, which are obtained either by applying either a truly random permutation or the pseudorandom one keyed by some key k_{i_1} unknown to $\mathcal{B}$. $\mathcal{B}$ sets each onion variant $O_{i_1,k} \in \vec{O}_{i_1}$ to be $((E_{i_1}, B_{i_1,1}, \ldots, B_{i_1,\ell-i_1+1}), C_{i_1}, (T_{i_1,k+1}, \ldots, T_{i_1,\ell_1-i_1-k+1}))$. $\mathcal{B}$ forms the outer layers $\vec{O}_{i_1-1}, \ldots, \vec{O}_1$ by following the $\mathsf{FormOnion}$ procedure, by building on $(H_{i_1}, C_{i_1}, S_{i_1})$. $\mathcal{B}$ sends O_1 to $\mathcal{A}$.
6. Once again, $\mathcal{A}$ is allowed to query onions. The first time $\mathcal{A}$ asks to have an onion $O_{i_1,k} \in \vec{O}_{i_1}$ peeled (or bruised), the challenger responds with its peeled version $O_{i_1+1,k}$ (resp. $O_{i_1+1,k+1}$). Whenever, $\mathcal{A}$ queries an onion $O = ((E_{i_1}, B), C, S) \notin \vec{O}_{i_1}$, $\mathcal{B}$ responds with $\perp$. For all other queries, $\mathcal{B}$ deals with them in the same way as before.
7. If $\mathcal{A}$ guesses Hybrid_2, $\mathcal{B}$ guesses that the challenge blocks are pseudorandom. Otherwise if $\mathcal{A}$ guesses Hybrid_3, $\mathcal{B}$ guesses that the blocks are truly random.

The reduction works because $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s, and the reduction runs in polynomial time. □

Lemma 7. Hybrid_5 *and* Hybrid_6 *are indistinguishable.*

Proof. The reduction is essentially the same as the proof that Hybrid_2 and Hybrid_3 are indistinguishable. The reduction differs only in step 6.

Assume that there exists a p.p.t. adversary $\mathcal{A}$ that can distinguish whether it is in Hybrid_2 or Hybrid_3 with non-negligible (in the security parameter) advantage. We construct the following reduction $\mathcal{B}$. $\mathcal{B}$ plays the role of the challenger for $\mathcal{A}$ and that of the adversary in the PRP security game against the challenger $\mathcal{C}$.

1. During setup, $\mathcal{B}$ interacts only with $\mathcal{A}$; it does not interact with $\mathcal{C}$. $\mathcal{A}$ sends the names of the honest parties P_m, P_g, P_r and the public portions $\mathsf{pk}(\mathsf{Bad})$ of the keys belonging to the adversarial parties to $\mathcal{B}$.
2. $\mathcal{B}$ generates the keys for the honest parties and sends the public portions of the generated keys to $\mathcal{A}$.
3. During the first query phase, $\mathcal{B}$ still interacts only with $\mathcal{A}$. Whenever $\mathcal{A}$ sends $\mathcal{B}$ a query to peel (resp. bruise) an onion O on behalf of an honest party P, $\mathcal{B}$ responds with the output of $\mathsf{PeelOnion}$ (resp. $\mathsf{BruiseOnion}$) on O and P's secret key.
4. During the challenge phase, $\mathcal{A}$ sends to $\mathcal{B}$ the onion parameters: the message m and the routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ such that $P_m = M_{i_1}$ and $P_g = G_{j=i_2-\ell_1}$.
5. $\mathcal{B}$ forms the inner onion layers $\vec{O}_\ell, \ldots, \vec{O}_{i_1+1}$ by following the $\mathsf{FormOnion}$ algorithm (on input: the dummy message content $\perp$ and the truncated path $(M_1, \ldots, M_{i_1}, \vec{\perp})$ and associated sequence $(y_1, \ldots, y_{i_1}, \vec{\perp})$ of metadata) but deviates from the algorithm at the ${i_1}^{th}$ layers $\vec{O}_{i_1}$. First, $\mathcal{B}$ forms the ciphertext E_{i_1} by encrypting the dummy key 0 under the public key of

M_{i_1}. Next, $\mathcal{B}$ sends to $\mathcal{C}$: the blocks $((M_{i_1+1}, E_{i_1+1}), B_{i_1+1,1}, \ldots, B_{i_1+1,\ell-i_1})$, the content C_{i_1+1}, and the $i_1 + 1^{st}$ layer T_{i_1+1} of the sepal for the first processor on the path. $\mathcal{C}$ replies with $(B_{i_1,1}, \ldots, B_{i_1,\ell-i_1+1}, C_{i_1}, T_{i_1})$, which are obtained either by applying either a truly random permutation or the pseudorandom one keyed by some key k_{i_1} unknown to $\mathcal{B}$. $\mathcal{B}$ sets each onion variant $O_{i_1,k} \in \vec{O}_{i_1}$ to be $((E_{i_1}, B_{i_1,1}, \ldots, B_{i_1,\ell-i_1+1}), C_{i_1}, (T_{i_1,k+1}, \ldots, T_{i_1,\ell_1-i_1-k+1}))$. $\mathcal{B}$ forms the outer layers $\vec{O}_{i_1-1}, \ldots, \vec{O}_1$ by following the $\mathsf{FormOnion}$ procedure, by building on $(H_{i_1}, C_{i_1}, S_{i_1})$. $\mathcal{B}$ sends O_1 to $\mathcal{A}$.

6. Once again, $\mathcal{A}$ is allowed to query onions. $\mathcal{B}$ handles the first query to peel or bruise an onion $O_{i_1,k} \in \vec{O}_{i_1}$ by running $\mathsf{FormOnion}$ on the message m, the routing path $(\vec{\bot}, M_{i_1+1}, \ldots, R)$, and the associated sequence of metadata $(\vec{\bot}, y_{i_1+1}, \ldots, y_{\ell-1})$; $\mathcal{B}$ replies with the onion $O_{i_1+1,k}$ for the $i_1 + 1^{st}$ processor on the routing path with k bruises, see Hybrid_5. Whenever, $\mathcal{A}$ queries an onion $O = ((E_{i_1}, B), C, S) \notin \vec{O}_{i_1}$, $\mathcal{B}$ responds with $\bot$. All other queries are handled in the same manner as before.
7. If $\mathcal{A}$ guesses Hybrid_5, $\mathcal{B}$ guesses that the challenge blocks are truly random. Otherwise if $\mathcal{A}$ guesses Hybrid_6, $\mathcal{B}$ guesses that the blocks are pseudorandom.

The reduction works because $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s, and the reduction runs in polynomial time. $\square$

Lemma 8. Hybrid_6 *and* Hybrid_7 *are indistinguishable.*

Proof. The reduction is essentially the same as the proof that Hybrid_0 and Hybrid_1 are indistinguishable. The reduction differs only in steps 5 and 6.

Assume that there exists a p.p.t. adversary $\mathcal{A}$ that can distinguish whether it is in Hybrid_6 or Hybrid_7 with non-negligible (in the security parameter) advantage. We construct the following reduction $\mathcal{B}$. ($\mathcal{B}$ is the challenger for $\mathcal{A}$, but the adversary in the CCA2 security game against a challenger $\mathcal{C}$.)

1. During setup, $\mathcal{A}$ sends the names of the honest parties P_m, P_g, and P_r, along with the public keys of the adversarial parties $\mathsf{pk}(\mathsf{Bad})$ to $\mathcal{B}$.
2. $\mathcal{B}$ generates key pairs for P_g and P_r; that is, $\mathcal{B}$ obtains $(\mathsf{pk}(P_g), \mathsf{sk}(P_g))$ by running $\mathsf{KeyGen}(P_g)$ and $(\mathsf{pk}(P_r), \mathsf{sk}(P_r))$ by running $\mathsf{KeyGen}(P_r)$. $\mathcal{C}$ supplies the public key $\mathsf{pk}(P_m)$ of P_m, which $\mathcal{B}$ forwards to $\mathcal{A}$ along with the public keys $\mathsf{pk}(P_g), \mathsf{pk}(P_r)$.
3. During the first query phase, whenever $\mathcal{A}$ sends an onion $O = ((E, B), C, S)$ to be peeled (resp. bruised) on behalf of P_m, $\mathcal{B}$ sends the ciphertext portion E to $\mathcal{C}$ to be decrypted. Once $\mathcal{C}$ replies with the layer key k, $\mathcal{B}$ completes the $\mathsf{PeelOnion}$ (resp. $\mathsf{BruiseOnion}$) algorithm on O and replies to $\mathcal{A}$ with the output. (For queries to peel or bruise on behalf of P_g and P_r, $\mathcal{B}$ just runs $\mathsf{PeelOnion}$ or $\mathsf{BruiseOnion}$.)
4. During the challenge phase, $\mathcal{A}$ sends to $\mathcal{B}$ the onion parameters: the message m and the routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ such that $P_m = M_{i_1}$ and $P_g = G_{j=i_2-\ell_1}$.
5. $\mathcal{B}$ picks the layer keys $k_1, \ldots, k_\ell$ and sends the challenge messages $m_0 = k_{i_1}$ and $m_1 = 0$ to $\mathcal{C}$. Once $\mathcal{B}$ obtains the ciphertext $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}(M_{i_1}, m_b))$ from $\mathcal{C}$, it forms the challenge onion O_1 by running $\mathsf{FormOnion}$ (on input: the dummy message content $\bot$ and the truncated path $(M_1, \ldots, M_{i_1}, \vec{\bot})$ and associated sequence $(y_1, \ldots, y_{i_1}, \vec{\bot})$ of metadata), except in forming the ${i_1}^{th}$ onion layer, $\mathcal{B}$ uses c_b in place of the ciphertext E_{i_1}. $\mathcal{B}$ sends O_1 to $\mathcal{A}$.
6. Once again, $\mathcal{A}$ is allowed to query onions. $\mathcal{B}$ handles the first query to peel or bruise an onion $O_{i_1,k} \in \vec{O}_{i_1}$ by running $\mathsf{FormOnion}$ on the message m, the routing path $(\vec{\bot}, M_{i_1+1}, \ldots, R)$, and the associated sequence of metadata $(\vec{\bot}, y_{i_1+1}, \ldots, y_{\ell-1})$; $\mathcal{B}$ replies with the onion $O_{i_1+1,k}$ for the $i_1 + 1^{st}$ processor on the routing path with k bruises, see Hybrid_5. All other queries are handled in the same manner as before.

7. If $\mathcal{A}$ guesses Hybrid_6, $\mathcal{B}$ guesses 1. Otherwise if $\mathcal{A}$ guesses Hybrid_7, $\mathcal{B}$ guesses 0.

The reduction works because $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s, and the reduction runs in polynomial time. $\square$

A.2 Reductions for $\mathsf{Hybrid}_8, \ldots, \mathsf{Hybrid}_{15}$

$\mathsf{Hybrid}_8, \ldots, \mathsf{Hybrid}_{15}$ convert the "middle layers" of the challenge onion in Hybrid_7 to those in Experiment 1. Thus, the reductions for these hybrids are essentially the same arguments as the reductions for $\mathsf{Hybrid}_0, \ldots, \mathsf{Hybrid}_7$.

A.3 Reductions for $\mathsf{Hybrid}_{16}, \ldots, \mathsf{Hybrid}_{18}$

$\mathsf{Hybrid}_{16}, \ldots, \mathsf{Hybrid}_{18}$ convert the "innermost layers" of the challenge onion in Hybrid_{15} to those in Experiment 1.

Lemma 9. *Hybrid_{15} and Hybrid_{16} are indistinguishable.*

Proof. The reduction is similar to the proof that Hybrid_6 and Hybrid_7 are indistinguishable. The main difference is that, here, the penultimate layer is pseudorandom or truly random as opposed to the i_1^{th} layer.

Assume that there exists a p.p.t. adversary $\mathcal{A}$ that can distinguish whether it is in Hybrid_{15} or Hybrid_{16} with non-negligible (in the security parameter) advantage. We construct the following reduction $\mathcal{B}$. $\mathcal{B}$ plays the role of the challenger for $\mathcal{A}$ and that of the adversary in the PRP security game against the challenger $\mathcal{C}$.

1. During setup, $\mathcal{B}$ interacts only with $\mathcal{A}$; it does not interact with $\mathcal{C}$. $\mathcal{A}$ sends the names of the honest parties P_m, P_g, P_r and the public portions $\mathsf{pk}(\mathsf{Bad})$ of the keys belonging to the adversarial parties to $\mathcal{B}$.
2. $\mathcal{B}$ generates the keys for the honest parties and sends the public portions of the generated keys to $\mathcal{A}$.
3. During the first query phase, $\mathcal{B}$ still interacts only with $\mathcal{A}$. Whenever $\mathcal{A}$ sends $\mathcal{B}$ a query to peel (resp. bruise) an onion O on behalf of an honest party P, $\mathcal{B}$ responds with the output of $\mathsf{PeelOnion}$ (resp. $\mathsf{BruiseOnion}$) on O and P's secret key.
4. During the challenge phase, $\mathcal{A}$ sends to $\mathcal{B}$ the onion parameters: the message m and the routing path $\vec{P} = (M_1, \ldots, M_{\ell_1}, G_1, \ldots, G_{\ell_2}, R)$ such that $P_m = M_{i_1}$ and $P_g = G_{j=i_2-\ell_1}$.
5. $\mathcal{B}$ forms the challenge onion O_1 by running $\mathsf{FormOnion}$ on the dummy message content $\perp$ and the truncated path $(M_1, \ldots, M_{i_1}, \vec{\perp})$ and associated sequence $(y_1, \ldots, y_{i_1}, \vec{\perp})$ of metadata, see Hybrid_4. $\mathcal{B}$ sends O_1 to $\mathcal{A}$.
6. Once again, $\mathcal{A}$ is allowed to query onions. $\mathcal{B}$ handles the first query to peel or bruise the challenge onion as follows:
   - $\mathcal{B}$ forms the outer layers $\vec{O}_1, \ldots, \vec{O}_{i_1}$ by following the $\mathsf{FormOnion}$ procedure, using $\perp$ as the message, $(M_1, \ldots, M_{i_1}, \vec{\perp})$ as the routing path, and $(y_1, \ldots, y_{i_1}, \vec{\perp})$ as the sequence of metadata.
   - $\mathcal{B}$ forms the middle layers $\vec{O}_{i_1+1}, \ldots, \vec{O}_{i_2}$ by following the $\mathsf{FormOnion}$ algorithm, using $\perp$ as the message, $(\vec{\perp}, M_{i_1+1}, \ldots, G_j, \vec{\perp})$ as the routing path, and $(\vec{\perp}, y_{i_1+1}, \ldots, y_{i_2}, \vec{\perp})$ as the sequence of metadata. To peel (resp. bruise) the i_i^{th} challenge onion, $\mathcal{B}$ returns the unbruised (resp. once bruised) version of the $i+1^{th}$ challenge onion. However, $\mathcal{B}$ keeps track of the total number of bruises that have accumulated thus far.
   - $\mathcal{B}$ deviates from the $\mathsf{FormOnion}$ algorithm in forming the inner layers $\vec{O}_{i_2+1}, \ldots, \vec{O}_{\ell-1}$: First, $\mathcal{B}$ forms the dummy sepal blocks $(T_{\ell,d+1}, \ldots, T_{\ell,\ell_1+1})$, the content C_ℓ, and the

ciphertext E_ℓ by following the algorithm and sends these blocks to $\mathcal{C}$. $\mathcal{C}$ replies with $(B_{\ell-1,1}, C_{\ell-1}, (T_{\ell-1,d+1}, \ldots, T_{\ell-1,\ell_1+1}))$, which are obtained either by applying either a truly random permutation or the pseudorandom one keyed by some key K unknown to $\mathcal{B}$. $\mathcal{B}$ forms the ciphertext $E_{\ell-1}$ by following the $\mathsf{FormOnion}$ procedure and sets each "too bruised" onion variant $O_{\ell-1,k} \in \vec{O}_{\ell-1}$ to be $((E_{\ell-1}, B_{\ell-1,1}), C_{\ell-1}, (T_{\ell-1,k+1}))$. $\mathcal{B}$ forms the "too bruised" layers in $\vec{O}_{\ell-2}, \ldots, \vec{O}_{i_2+1}$ by following the $\mathsf{FormOnion}$ procedure, by building on the $\ell - 1^{st}$ layers, and returns the onion layer with the correct number of bruises.

- Whenever, $\mathcal{A}$ queries an onion $O = ((E_{i_1}, B), C, S) \notin \vec{O}_{i_1}$, $\mathcal{B}$ responds with $\perp$.

Whenever, $\mathcal{A}$ queries an onion $O = ((E_{i_2}, B), C, S) \notin \vec{O}_{i_2}$, $\mathcal{B}$ responds with $\perp$. All other queries are handled in the same manner as before.

7. If $\mathcal{A}$ guesses Hybrid_{15}, $\mathcal{B}$ guesses pseudorandom. Otherwise if $\mathcal{A}$ guesses Hybrid_{16}, $\mathcal{B}$ guesses truly random.

The reduction works because $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s, and the reduction runs in polynomial time. $\square$

Finally, the proof that Hybrid_{17} and Hybrid_{18} are indistinguishable is the same argument as above except that, in step 6, all of the sepal blocks are dummy blocks, and the onion returned by $\mathcal{B}$ (on behalf of G_j) is unbruised.