Secure k-anonymous Communication Protocols with Misbehavior Detection

Greg Benjamin
gregben@cs.umd.edu

Christopher Hayden
cmhayden@cs.umd.edu

Department of Computer Science
University of Maryland, College Park

Abstract

Anonymity, although often a desired property in a communications network, is still very much an open problem, as existing protocols tend to suffer from unreasonable overhead or pathological vulnerability to attack and abuse by malicious users. In this paper, we explore k-anonymity, a refinement of the problem in which participants attempt to "hide in a crowd" of size k, such that no initiator of an action may be pinpointed as such with any certainty greater than $1/k$. We survey several existing protocols and provide two of our own, which we believe provide unique solutions to some of the more difficult problems associated with k-anonymous communication, particularly the issue of identifying and removing malicious participants. We also analyze the proposed protocols in terms of several known attacks against anonymizing overlays, provable guarantees of k-anonymity, efficiency, and overhead.

1 Introduction

Secure correspondence is a desirable property in a communications network. Three capabilities such a network should provide are authentication, obfuscation, and anonymity. The first two of these are heavily researched and are largely solved: public key infrastructures (PKIs) allow for authentication, while obfuscation is provided by data encryption. Anonymity, however, is still very much an open problem. In order to reliably send messages between communicants, it is generally necessary to know their identities. Furthermore, existing protocols suffer from one of two deficiencies: unreasonable overhead, in the form of persistent noise[8], topological constraints[1, 2], or throughput limitations[2]; or pathological vulnerability to attack and abuse by malicious users[3, 4].

In this paper, we restrict ourselves to considering a refinement of the anonymity problem: k-anonymity[12]. In k-anonymity, the goal is to "hide in a crowd" - that is, to guarantee that the cardinality of the set of possible initiators of an action cannot be reduced below k. We present two protocols that provide sender k-anonymity, which we call Protocol-1 and Protocol-2.

Protocol-1 borrows ideas from both Crowds[6] and Tor[13]. It uses recursive path extension in order to establish a fixed ad-hoc tunnel through a set of sender-selected peers. Communications and responses are layer encrypted and routed along the peers in the path. A mechanism for misbehavior detection allows the sender to detect and expel Byzantine peers along the path without leaking information about its identity. As a result, Protocol-1 is robust against many of the attacks that plague Crowds[6].

Protocol-2 provides stronger anonymity guarantees than Protocol-1 at the cost of requiring peers to organize themselves into fixed logical rings, similar to those seen in [2]. Communications are layer-encrypted and routed to a sender-selected exit node. The exit node engages the recipient, encrypts its response, and routes it along the ring back to the sender. We use a pairwise key exchange during construction of the overlay in order to eliminate the need for a PKI. Like Protocol-1, we use a misbehavior detection mechanism to allow participants to identify and expel Byzantine peers. The mechanism is particularly novel because Byzantine agreement is not required to verify accusations. Instead, accusations are treated as self-sacrificial actions by which the accusing peer sacrifices itself in order to eliminate a Byzantine peer. By eliminating both the accuser and the accused, we provide a disincentive for misbehaving peers to accuse innocent peers and we guarantee that colluding peers are not able to acquire control of the ring.

We provide an analysis of both protocols in terms of resilience to a variety of known attacks against anonymizing overlays, as well as a few that we uncovered during the process of designing the protocols. Such attacks tend to take one of two possible avenues: they either seek to deanonymize the partic-

ipants of such an overlay by linking a sender with a specific query or message, or they attempt to deny service entirely to said participants. We also examine the guarantees of provable k-anonymity provided by the protocols, using Chernoff bounds to compute the probability of being in an anonymous set of cardinality k, given that some fraction f of the participating nodes are malicious. Finally, we explore the usefulness of the protocols in terms of efficiency, network resource usage, and computation and storage requirements for the participating nodes.

The remainder of the paper is structured as follows. In Section 2, we summarize several existing protocols and note their limitations. Section 3 formalizes the problem, presents our goals, and introduces our adversarial assumptions. We give a detailed description of our protocols in section 4 and we provide an analysis of security and robustness in section 5. In section 6, we conclude and discuss avenues for future work.

2 Background

2.0.1 Terminology

Three types of anonymity have been defined in previous work[24, 6, 2]: *sender anonymity*, *recipient anonymity*, and *relationship anonymity*. We adopt the definitions of [24]. *Sender anonymity* implies that is impossible to link any message to a particular sender or to link any sender to a particular message. *Receiver anonymity* similarly implies that it is impossible to link any message to a particular recipient or to link and recipient with a particular message. *Relationship anonymity* is a weaker version of anonymity. It implies that it is impossible to establish that a particular sender and a particular recipient are engaged in a communication with one another, though it may be clear that each is engaged in communication with *someone*. A system that has all three of the above properties can be considered *fully anonymous*[2].

k-anonymity provides a weaker guarantee on anonymity than full anonymity. In [12], k-anonymity is defined as the inability to distinguish one individual's information from the information of at least $k-1$ other individuals. Ahn, Bortz, and Hopper adapt this definition to the domain of anonymous communications[25]: sender (recipient) k-anonymity is the inability to distinguish the sender (recipient) of a communication from at least $k-1$ other honest participants. Wang, Ning, and Reeves suggest an alternative definition[2]: the sender (recipient) of a communication cannot be linked to that communication with probability greater than $\frac{1}{k}$.

Anonymous communication systems distinguish between unlinkability and unobservability[29]. Unlinkability means that two items of interest in a system are no more or less likely to be related after an attacker observes them than they are based solely on *a priori* information. Unobservability means items of interest are indistinguishable from other items of the same type. In terms of anonymous systems, unlinkability implies that an attacker cannot associate a user with a message even if he observes both, while unobservability implies that an attacker cannot determine when a user is using the system or when a communication is occurring.

2.0.2 Previous Work

Much of the work on anonymous communication systems can be traced back to Chaum's seminal efforts: mixnet[18] and DC-net[19]. In mixnet, senders encrypt their messages with the public key of a trusted server, called a mix, and forward the messages to the server. The server collects a batch of encrypted messages, decrypts them, and forwards them to the recipients. An attacker cannot link any decrypted message output by the mix with any encrypted message sent to the mix. Mixminion is an example of a system that uses this approach[14]. DC-net[19] provides provably secure multiparty communication without the need for a trusted third party. However, DC-net suffers from collisions: only a single party may communicate during any time-slot. DC-net also imposes a heavy burden on the underlying network, requiring $O(N^2)$ protocol messages for each transmitted communication.

Tor is a circuit-based anonymous communication service that allows users to establish a tunnel through a closed set of onion routers [13]. Senders negotiate a session key with each server on the circuit to wrap messages in layered encryption. This enables the system to provide perfect forward secrecy and integrity checking. MorphMix[10] takes a similar approach. However, these systems are difficult to scale and they are brittle in the face of node failures, which makes predecessor attacks possible[17]. Murdoch and Danezis demonstrate how the anonymity provided by Tor can be compromised using traffic analysis[4].

Several proposals attempt to harness peer-to-peer networks to address scalability problems, including Crowds[6], Hordes[7], Tarzan[9], Herbivore[11], and Salsa[27]. These systems suffer from intrinsic issues in peer-to-peer systems, such as high node churn. A more recent proposal by Li et al. addresses node churn by exploring multiple pathways through a set of commutative relay groups[28]. When node failures

occur, "path hopping" is used to switch paths without altering the path prefix. Despite this, the system is still vulnerable to statistical attacks[4] and predecessor attacks[17].

Wang, Ning, and Reeves provide k-anonymity by arranging nodes into distinct logical rings of size k. [2]. Within each ring, a round-based anonymous communication mechanism uses message batching and a reverse hash chain to make communications indistinguishable and to avoid the collision issues that plague DC-Net[19].

The above systems above provide unlinkability, but not unobservability. Danezis and Wittneben argue that only unobservability is sufficient to guarantee anonymity because unlinkability is falls victim to information leaked by third parties[16]. Anonymous systems that provide unobservability include Nonesuch[26] and $\mathcal{P}^5$[8]. The former uses steganography to embed messages into a subset of common transactions, while the latter floods the network with random noise when communication is not occurring.

2.0.3 Reputation Systems

Several works propose using a reputation system in order to tolerate malicious users in anonymity protocols.[23, 15, 21, 22]. However, these systems from the opposing requirements of an anonymity system and a reputation system. Both [21] and [23] use signed receipts in order to bind participants to their agreements, but they are brittle because they require witness sets in order to verify receipts and take action against deviants. The authors of [15] propose a system using e-cash that allows users to change pseudonyms without sacrificing their reputations. Unfortunately this system relies on a central server to function as the "bank", the database that serves and maintains participant reputations, and therefore suffers from a single point of failure and a lack of scalability.

3 Problem and Assumptions

We formalize the problem as follows: a sender s wants to be able to send queries to some receiver r and receive responses to said queries. However, these queries are sensitive in nature; s does not want any other party, including r, to be able to map any query he originated back to him. Obviously, this goal is somewhat at odds with the ability to receive responses to queries, so we will restrict our problem to providing the property of *sender k-anonymity*: for any query q which s originated, no other party should be able to map q back to a set of potential senders any smaller than k in size.

Our solution for this problem is for the node s to participate in an overlay of other nodes with the express aim of providing such k-anonymity. Within this overlay, we provide a pair of protocols for the sending of anonymous queries and for the return of response messages from the recipient. We claim that our protocols provide k-anonymity to the sender and also are robust to any malicious attempts to expose senders or deny service to participating nodes.

In our protocols, we assume the following adversarial model:

- Some fraction $f < \frac{1}{2}$ of the nodes participating in the protocol are Byzantine and malicious. All malicious nodes are colluding with each other.
- The targeted recipient r may be malicious and colluding.
- There may also be a malicious entity locally observing traffic on some fraction of the links in the network. We assume such an entity is colluding as well, but that the fraction of links he can observe is proportional to f - he cannot globally observe the entire network.
- Malicious nodes are reasonably bounded in both computational power and storage space. This guarantees us that today's standard cryptographic hardness assumptions do in fact hold - malicious nodes cannot reverse hashes or decrypt without a key, for example.
- All queries from the same source are in some sense "linkable" by their content, and malicious nodes with access to the plain-text queries are able to link them, whether by statistical means or otherwise.

We also make the following simplifying assumptions:

- Each node in the overlay has a single, static IP address. The IP addresses of all nodes are known to all other nodes in the overlay.
- We also assume the IP protocol to be secure. That is, if two nodes can exchange messages and participate in an interactive protocol, we assume that they are in fact communicating with each other.
- We assume that nodes have some way of learning about other nodes who are participating in the protocol - i.e., each node knows about a "neighborhood set" of other participating nodes

We make these assumptions because it slightly simplifies the problem, and also because we believe the issues raised by these assumptions, particularly those of securing IP, to be orthogonal in nature to the k-anonymity we are attempting to provide.

4 Protocol Specifications

4.1 Protocol-1

Protocol-1 seeks to provide anonymity by constructing a "forwarding tunnel", similarly to the Crowds and Tor protocols. The idea is that only the sender should know the structure of the tunnel and which nodes are participating; all other nodes should only know their local predecessor and successor. To accomplish this, the tunnel must be constructed iteratively, so that all messages to a particular node come through his predecessor. End-to-end encryption is used to confirm that when the tunnel is extended, the new node is indeed the one selected by the sender. Here, we assume the existence of a trusted PKI, although we also discuss possibilities for designing a key exchange, and the implications thereof.

4.1.1 Setup

For some node s to communicate anonymously with a target r, s first needs to construct his anonymizing tunnel. First, s will pick a set of participating nodes from his neighbor set, and order them in a path from himself to an appointed "exit node", like so:

$$path = \{s, n_1, n_2, n_3, \ldots, n_{exit}\}$$

s will then connect to the first node n_1 by TCP and send a *setuplink* message. n_1 will respond with an acknowledging *linksetup*, confirming that n_1 expects to receive and forward messages from s. s and n_1 should then perform a handshake to confirm public keys.

Once the link from s to n_1 is established, s can extend his tunnel to each node in the path via the following inductive method:

- s uses the secure sending protocol described below to send an $extendpath(n_i, n_{i+1})$ message to node n_i.
- n_i connects to the provided n_{i+1} via TCP and sends a *setuplink*.
- n_{i+1} responds with a *linksetup* and keys are confirmed.
- n_i uses the secure response protocol below to send an acknowledging $pathextended(n_i, n_{i+1})$ back to s.

The inductive process ends once the tunnel is extended to n_{exit}.

4.1.2 Sending a Message

To send a query q to target r through the tunnel, s will select an ephemeral key $ephk$ for the return and will compose the message $M = (q, r, ephk)$. s will then layer-encrypt M with the public keys of all nodes on the path, in reverse order, as follows:

$$CT = enc_{pubk_1}(enc_{pubk_2}(\ldots enc_{pubk_{exit}}(M)\ldots))$$

Finally, s will generate a $flowid$ and will send $(flowid, CT)$ to n_1, his successor on the path, and will wait for a signed receipt from n_1, as described next.

4.1.3 Receiving and Forwarding

When n_i receives a message $(flowid, CT)$ from his predecessor n_{i-1}, he must immediately compute a signed receipt for the message he received and give it to n_{i-1}. n_i will compute

$$R = sign_{privk_i}(flowid, CT)$$

and send it to n_{i-1}, who will verify R using $pubk_i$. n_{i-1} will then cache the receipt as evidence that he faithfully forwarded CT to his successor. If no receipt is provided, n_{i-1} will transmit an accusation message along the return path, as described below.

n_i will then remove a layer of encryption from CT using his private key:

$$\begin{aligned} CT' &= dec_{privk_i}(enc_{pubk_i}(\ldots enc_{pubk_{exit}}(M)\ldots)) \\ &= enc_{pubk_{i+1}}(\ldots enc_{pubk_{exit}}(M)\ldots) \end{aligned}$$

If n_i is actually the exit node, $CT' = M$, and so he can then break the message into its pieces and communicate the query q to receiver r via TCP.

Otherwise, n_i will compute and store a transformation from $flowid$ into a new $flowid'$. (Having a $flowid$ is necessary in order to process responses and receipts correctly, but the $flowid$ must change from hop to hop in order to preserve untraceability along the path.) He will then forward $(flowid', CT')$ on to his successor n_{i+1}, wait for a signed receipt, and verify and cache it, as before.

4.1.4 Handling Responses

Once n_exit receives a response $resp$ from r, he is responsible to return that response back up the tunnel to s. To that effect, n_exit will use the ephemeral key $ephk$ extracted from the original message he received and compute $CT = enc_{ephk}(resp)$. $(flowid', CT)$ will be sent to his predecessor, n_{exit-1}, who will generate and sign a receipt, as before. The predecessor will map $flowid'$ back to the previous $flowid$ and forward back to his predecessor. This process continues iteratively: n_{i+1} forwards to n_i, and so on, until the response reaches s. The response protocol is essentially precisely the same as the forward- sending protocol, only in reverse, with the major change that there is no layered encryption, so nodes only forward the response without decrypting it.

4.1.5 Detection of Malicious Nodes

If at any time node n_i forwards to n_{i+1} and does not receive a valid receipt in return, n_i should use the return protocol to send a signed accusation message

$$A = sign_{privk_i}(flowid, n_{i+1})$$

When the sender s receieves and verifies this message, he knows that n_{i+1} is being accused of not generating a receipt, and so he may truncate his sending tunnel above n_{i+1} and iteratively rebuild from n_i.

It is worth noting that n_i may accuse his successor at any time, even if n_{i+1} did generate a valid receipt. To counteract this, s should only accept some threshold number t of accusations from any particular node. If the number of accusations from any n_i exceeds t, s should assume n_i is acting maliciously, truncate the path above n_i, and rebuild from n_{i-1}. It is also possible for a malicious node further up the path to drop the accusation message; this case will be discussed momentarily.

If at some point s times out while waiting to a response from r, he should assume that there is some fault in his sending tunnel. At this point, s should send a $probepath(flowid)$ message in the clear through the sending tunnel. Each node receiving the message knows to transform the $flowid$, forward the message on, and wait for a response. Once the message reaches n_{exit}, he will return a $receiptchain(endofpath)$ message to his predecessor. Each node in the path will now append their signed forwarding receipt to the chain and forward it to their predecessor. In this way, the sender should receive a receipt chain consisting of valid signed forwarding receipts for the entire path, and can verify them using the public keys of the nodes on the path.

If a node n_i on the path cannot produce a receipt, then it must be that he himself did not forward - otherwise he would have previously issued an accusation. If n_i inserts garbage into the receipt chain, the sender will be unable to verify his receipt and will detect him. Otherwise, he can simply not forward the receipt chain, in which case his predecessor n_{i-1} will time out waiting for the receipt chain and simply return a new chain that ends at n_{i-1}. In either case, the sender knows that n_i is malicious and may truncate the tunnel at n_i and build a new tunnel suffix from n_{i-1}.

The other possible case is that a node n_i drops the receipt chain from his successor and returns one ending at n_i, thus framing n_{i+1} for not having produced a receipt. In this case, s will again build a new tunnel suffix from n_i, but will also keep track of the number of times that he has done so. Once that number of times exceeds the threshold t, s knows that it is likely that n_i is framing his successor, and so should drop n_i from the tunnel and rebuild from n_{i-1} instead.

Finally, if the sending path appears good, the same process may be used to probe for receipts along the return path, and non-forwarding nodes may be cut off in the same manner. In this case, however, nodes that never received the packet will append a signed message $neverreceived(flowid)$ to the receipt chain, since they do not have any valid receipts to append. The timeout process works similarly, and so the sender should receive a chain beginning with $neverreceived$ messages and terminating with some number of legitimate receipts. If a node cannot produce a receipt, it is probable that he will append a $neverreceived$ and attempt to frame his successor by removing his receipt from the chain. Here, the same threshold rule applies - after the path is truncated immediately below any particular node t times, that node itself will be truncated next, since this is considered strong evidence of that node maliciously framing his successors.

This entire misbehavior-detection procedure operates on the principle of attempting to preserve a prefix of good nodes participating in the tunnel. Malicious nodes can never frame a node above them, only the node below them. Furthermore, malicious nodes may only tamper with messages coming from the path below them, by just not forwarding receipts or accusations or some such, but never with messages above them on the path. This guarantees that any path prefix of all-good nodes, even only a trivial prefix of just the sender, cannot be compromised or made to look malicious. This is important, because if it were not the case, a malicious node immediately below the sender could make the sender look malicious, and

therefore any path truncation that did not remove the sender would leak information to the malicious party, since the sender obviously does not wish to be removed from his own path. Furthermore, malicious nodes that do tamper with the nodes below them can only do so some threshold number of times before being truncated themselves. Given enough time, any node messing around will be caught and removed from the path, and so the sender can improve his path by small iterations until he has a tunnel that works consistently.

4.1.6 Relaxing the PKI Assumption

Optimally, we'd like to be able to relax the assumption that requires a PKI. We've experimented with using the partially-constructed tunnel to create and exchange keys via a Diffie-Hellman handshake, but this is not a perfect solution. For one thing, every node's predecessor and successor also needs to know his public key in order to verify receipts, in addition to the sender constructing the tunnel. Furthermore, this opens the sender up to a man-in-the-middle attack, whereby malicious nodes may subvert and redirect the key exchange to another node then the one specified by the sender. In this way, the entire path may be hijacked. It's not entirely clear what the implications of this are, but if the first node in the path is malicious and the entire path is hijacked, we essentially end up in the same situation described in the Crowds paper[6]. In this case, the malicious nodes know that the first node's predecessor is the sender with some bounded, but relatively high probability.

As much as we would like to be able to remove the PKI from our protocol, we have not yet succeeded in finding a workaround to the above issues, particularly the man-in-the-middle attack.

4.2 Protocol-2

Protocol-2 is based on the idea of composing participating nodes into a DHT-like ring, where each node has a unique predecessor and successor. Messages are forwarded around the ring from along a path from the sender to an appointed "exit node", who will communicate with the target r via TCP. Such messages are encrypted using *layered encryption* with secret keys so that they are untraceable as they move along the ring. Nodes will also send probe packets from time to time, both to ensure the integrity of the ring, and to provide some level of noise in the overlay to protect against our local passive eavesdropper. Finally, because the ring is singly-linked, it is possible to for malicious nodes to simply drop packets and thereby disrupt communications for all participating nodes. Therefore, we provide a method whereby good nodes can figure out which nodes are misbehaving and evict them from the ring, thereby restoring integrity to the overlay.

4.2.1 Setup

The setup phase begins when some initiating node i chooses to create an anonymizing overlay. Node i will select the following parameters:

- n, the size of the overlay
- G, a cyclic group with prime order p and generator g
- b_{min}, the overlay's bandwidth lower bound (the rate at which probe traffic should be sent to keep the ring somewhat noisy)
- b_{max}, the overlay's bandwidth cap (the maximum rate at which a participating node is permitted to send data to his successor)

The initiator i then sends an invitation

$$I = (n, G = \{g, p\}, b_{min}, b_{max})$$

to some number $m \geq n - 1$ of other nodes from his neighborhood set and waits for a response indicating that they wish to participate. Once the requisite $n - 1$ affirmative responses have been gathered, he generates a random permutation

$$p = \{n_1, n_2, n_3, \ldots n_n\}$$

and sends each node a setup packet containing p. This serves to inform all participating nodes which other nodes will be in the overlay and establishes the ordering that forms them into a ring.

At this point, nodes should exchange keys and start sending *probe* messages out along the ring at rate b_{min}, via the methods discussed below.

4.2.2 Key Exchange

In order for the layered encryption scheme to work, each node needs to share a secret key with each other node on the ring. To accomplish this, we have a key exchange phase directly after the setup. Each node n_i picks some element $a \in Z_p$ and computes $g^a \in G$, which is then sent to all other nodes in the ring. Receiving nodes will store this g^a and follow the Diffie-Hellman protocol, picking an element $b \in Z_p$ and computing g^b and g^{ab}, which will become the shared secret key.

At this point, the receiver will confirm the key exchange by generating a random nonce r and sending n_i the message

$$M = (g^b, enc_{g^{ab}}(r))$$

n_i now knows a and g^b, so he can compute g^{ab}, decrypt r, and send it back to the challenger to confirm the shared key.

At this point, the challenger may discard his value of b, but he knows that n_i is bound to the value g^a. He can generate a message-specific session key for n_i at any time by picking a $c \in Z_p$, computing g^c and the key g^{ac}, and including g^c along with any encrypted packet sent to n_i.

The ring is now ready to send communications. All nodes should start sending probe messages with some frequency b_{min}.

4.2.3 Sending a Message

Suppose that a node wishes to anonymously send query q to a receiver r via the overlay. The sending node will select an "exit node" n_{exit} and compute the path $n_1, n_2, n_3, \ldots n_{exit}$ from himself to the exit node. Because of the key exchange, the sender knows a unique g^{a_i} for each node n_i on the path. He will generate a value $c \in Z_p$ and compute the shared key $g^{a_i c}$ for each node on the path. The sender will then append an all-zero $flag$ and the address of receiver r to the query q and encrypt the query as follows:

$$CT = enc_{g^{a_1 c}}(enc_{g^{a_2 c}}(\ldots enc_{g^{a_{exit} c}}(flag||q||r)\ldots))$$

Using this layered encryption scheme, the contents of the message and targeted recipient may only be known to the exit node, and only if the message has appropriately been forwarded through each node along the path.

Finally, the sender composes the message packet

$$M = (g^c, CT)$$

and sends it to n_1, the first node on the path.

At this point, the sender will await a signed receipt from n_i signifying that M was received, as discussed below.

4.2.4 Receiving and Forwarding

Nodes in the overlay are required to receive packets at a rate no greater than b_{max}. If a node receives packets from his predecessor in excess of this bandwidth cap, those packets will be dropped.

When a node n_i receives the message $M = (g^c, CT)$ from his predecessor, he will first generate and sign a receipt for the message:

$$R = sign_{a_i}(M)$$

This receipt R is then sent back to his predecessor, who caches it as evidence that he faithfully forwarded the message he was sent.

Next, n_i will use his a_i and g^c to compute $g^{a_i c}$, and will then strip off one layer of the encryption on CT:

$$\begin{aligned} CT' &= dec_{g^{a_i c}}(enc_{g^{a_i c}}(\ldots enc_{g^{a_{exit} c}}(flag||q||r)\ldots)) \\ &= enc_{g^{a_{i+1} c}}(\ldots enc_{g^{a_{exit} c}}(flag||q||r)\ldots) \end{aligned}$$

If the $flag$ field of CT' comes out to all zeroes, n_i knows that he was the intended exit node for this packet. He then splits CT' into q and r and sends the query to the recipient via TCP.

Alternatively, if $flag$ is nonzero, n_i needs to forward CT' on to his successor. However, he cannot violate the bandwidth cap of b_{max}. If sending a new message would cause him to surpass that bandwidth cap, he must wait until enough time has passed that he can safely send again. At that point, he will compose the message

$$M = (g^c, CT')$$

and send M to n_{i+1}, the next node in the path. Again, n_i will await a signed receipt from n_{i+1}, verify it using the known $g^{a_{i+1}}$, and cache it as evidence of his own good behavior.

4.2.5 Handling Responses

After n_{exit} receives a response message $resp$ from the recipient r via TCP, he is responsible to return the message to the original sender s. However, since he does not know the identity of the sender, he must relay $resp$ to all nodes in the ring. To do so, the exit node retains the g^c used to encrypt the original query q. He computes an *ephemeral key* $g^{a_{exit} c}$ for the response using his own a_{exit} value, and encrypts the response $resp$ accordingly. Finally, he composes the message

$$M = (enc_{g^{a_{exit} c}}(resp))$$

and sends M to his successor, who generates a receipt and forwards M along the ring. Each node in the ring receives M, generates a receipt, and forwards in turn, including the original sender. Eventually, M returns to n_{exit}, who recognizes it as the message he sent and does not forward again. It should be noted that response messages must be recognizable from query messages, since queries need to be layer-decrypted before forwarding, but responses must not be.

In this way, every node in the ring receives the message M, but since only the original sender s knows

the corresponding value of c, only s can decrypt this message and access the plain-text response.

(Note: In the case that the TCP communication with r is unsuccessful and times out, the exit node should generate a timeout message, encrypt that as the response, and return it as before.)

4.2.6 Probe Messages

Nodes should be sending probe messages out at a fixed rate b_{min} in order to provide a low level of background noise and insure the integrity of the ring. To send a probe message, the sender generates a random nonce r and composes a message containing an all-1s $flag$, $q = r$, and a blank receiver field. The message is layer-encrypted and forwarded as usual.

When the chosen exit node receives the message, he will decrypt the $flag$ field back to all-1s, which marks the message as a probe. The exit node will then immediately use the return protocol to send a response message to the probe containing nonce r. When the sender receives the response, he can check r to ensure that the probe was forwarded around the ring faithfully. If the response times out or r is tampered with, the sender can then challenge the nodes in the ring to produce their receipts, as discussed below.

4.2.7 Accusation of Malicious Nodes

There are three cases in which a participating node n_i may accuse another node n_j of malicious behavior. The first is when n_i sends a message to his successor n_j, and n_j does not generate a valid signed receipt for the message. In this case, n_j is known to be not following the protocol. n_i broadcasts a signed accusation message

$$A = sign_{a_i}(n_i, n_j)$$

to all the other nodes in the ring, who immediately verify the accusation with g^{a_i} and remove both n_i and n_j from the permutation of participating nodes used to encode the ring. In effect, the good node n_i sacrifices himself to remove the malicious node n_j from the overlay. All other nodes update their successor and predecessor pointers accordingly.

However, it is still possible for a bad node to generate the receipt, but just refuse to forward the message on to his predecessor. In this case, the sender s of the original message will never receive a response, but will instead timeout. At this point, the sender should select a new c and exit node n_{exit}, re-encrypt the query, and try again. If several such attempts time out, then there is a strong chance that some node along the path has refused to forward the message.

At this point, the sender will generate a probe message p and send it along the faulty path. If the probe is unsuccessful, the sender may then directly contact each node in the path, starting with his successor. He deanonymizes himself as the sender of the probe and challenges each node to produce a signed receipt for the message he apparently received and forwarded. If any node n_i fails to produce such a receipt, that node must be malicious, or else he would have accused his successor before. Therefore, the sender s may broadcast the signed accusation

$$A = sign_{a_s}(s, n_i)$$

which effectively removes both s and n_i from the overlay. If all nodes produce valid receipts for the forward path, s can compute what the correct response message should have looked like and challenge nodes to produce receipts for the response message by the same process.

Finally, a node n_i may accuse his predecessor of malicious behavior if he is receiving traffic at a rate higher than b_{max}. This indicates that the predecessor is violating the protocol and potentially trying to flood the ring with traffic as a denial-of-service attack (see section 5). n_i will broadcast the message

$$A = sign_{a_i}(n_i, n_{i-1})$$

thus removing both himself and his predecessor n_{i-1} from the overlay.

It is worth noting that malicious nodes have no incentive to falsely accuse good nodes of misbehavior, as they themselves will be evicted from the ring for doing so. In fact, for a bad node to do so is actually good for the integrity of the ring, given our assumption that $f < 1/2$. Furthermore, the signature on the accusation prevents malicious nodes from posing as other nodes and issuing accusations to get them evicted from the ring.

5 Analysis

We begin by considering the security of our protocols against known attacks on anonymity protocols. Then we show how participants may achieve k-anonymity with high probability in each of protocols. Finally, we discuss the efficiency of our protocols, both in terms of the overhead in network communications and in terms of the computation and storage burden placed on participants.

5.1 Resilience to Known Attacks

5.1.1 Intersection

If an adversary identifies linkable queries that originate from distinct groups for which the members are known, then he can reduce the number of possible senders by intersecting the members of the group. Specifically, suppose linkable queries $q_1, \ldots, q_j$ are attributable to some member of groups $G_1, \ldots, G_j$ with sizes $n_1, \ldots, n_j$, respectively. Then the originator of the queries must be in the intersection of the groups given by $\hat{G} = \cap_i G_i$. If the adversary is able to collect a sufficient number of linkable queries then the cardinality of the intersection will go to one and the originator will be identified.

There are two types of intersection attacks: short-term and long-term[3]. We discuss them separately.

Short-term Intersection. In a short-term intersection attack, we assume the adversary is able to observe the traffic on some subset of the links in the network. By noting when links are active and using information about how long it takes queries to traverse the network, the adversary may be able to identify the sets of possible senders for some number of linkable queries. By intersecting the sets of possible senders, the adversary is eventually able to identify with a high level of certainty the node that originated the queries[3].

If we were to assume that there are no local observers, both of our protocols would be trivially immune to this attack. However, if we allow an adversary to be a local observer then the protocols become vulnerable. Specifically, the adversary knows when a node issues a query because the outbound traffic will be greater than the inbound traffic. If the adversary is able to link queries that emerge from the network some fixed amount of time after the node issues them, then the adversary will eventually be able to identify the node as the originator with high probability. Protocol-1 is extremely succeptible to such an attack; the background noise generated by probe traffic in Protocol-2 makes the attack more difficult, but not impossible.

If nodes participate in multiple paths, we can make this attack more difficult by introducing random delays as packets are forwarded through the network. This flattens the distribution of the times taken by a node's queries to exit the network, which means the adversary needs to see many more packets before he can identify the node with high probability. Obviously Protocol-2 will benefit from random delays since all of the nodes in an overlay are part of the same path. Protocol-1 will benefit from random delays whenever the expected number of times that a node appears on all paths is greater than one.

Suppose a sender elects to construct a path of length k. Assume it samples the k nodes to appear on path uniformly at random from the set of all participating nodes and that the sampling is done with replacement. Then the number of times any given node can expect to appear on the sender's path is given by

$$\sum_{i=1}^{k} i \binom{k}{i} \left(\frac{1}{n}\right)^i \left(\frac{n-1}{n}\right)^{(n-i)} = \frac{k}{n}$$

where n is the number of nodes participating in the protocol. If we now assume k is sampled from some distribution over path lengths then the number of times a given node can expect to appear in the sender's path is given by

$$\sum_k \frac{k}{n} \Pr(k) = \frac{1}{n} E[k].$$

If each node constructs exactly one path, then the total number of times a given node can expect to appear on all paths, including its own, is given by

$$1 + n\frac{1}{n}E[k] = 1 + E[k].$$

This indicates that the number of times each node expects to appear on all paths depends only on the distribution from which the path lengths are sampled and not on the number of nodes participating in the protocol.

Long-term Intersection. In a long-term intersection attack, an adversary links queries across multiple sessions. By intersecting the sets of nodes participating in the sessions, the adversary is able to eventually identify with a high level certainty the node that originated the queries[3].

Both of our protocols are susceptible to long-term intersection attacks. However, our use of structured overlays in Protocol-2 means that its vulnerability is more acute. In anonymity protocols that use structured overlays consisting of subsets of the participating nodes, an adversary may be able to accelerate the rate at which he acquires information by using Sybils[5] to increase churn. Specifically, if the Sybils make up a large part of such an overlay, then they may be able to effect dissolution of the overlay by rapidly departing, thereby forcing legitimate nodes to join a new overlay. If the adversary identifies linkable queries exiting multiple distinct overlays, he may intersect the sets of nodes participating in those overlays to reduce the number of possible originators, eventually identifying the true originator with high probability.

5.1.2 Predecessor Attack

The idea underlying the predecessor attack[17] is that if a node appears on multiple paths of linkable queries, then that node is more likely to be the originator of the queries than any other node. The more such paths the node appears on, the higher this probability becomes. A malicious node may attempt to take advantage of this by effecting a change in the prefix of a path on which it appears. Since the originator of the queries along that path must appear in the newly constructed prefix, an adversary may be able to pinpoint him. Note that this attack subsumes fail-stop attacks, where a malicious node behaves unresponsively or refuses to forward packets.

Both of the protocols we present are resilient to this attack because a malicious node cannot cause a change to the prefix of a path on which it participates. In each protocol the sender uses misbehavior detection to identify the malicious node. In Protocol-1 the sender reconstructs the path from the predecessor of the malicious node, which modifies the suffix of the path but leaves the prefix intact. In Protocol-2 the sender issues an accusation whereby both he and the malicious node are removed from the ring. The remainder of the ring is static, so a malicious node cannot effect a change in either the prefix or the suffix of the path.

5.1.3 Invalid Forwarding

Byzantine nodes may attempt to effect a change in the prefix of a path by refusing to forward packets or my modifying them while in transit. If such an effort is successful, an adversary may be able to use the predecessor attack[17], as described above, in order to identify the sender. Even if the adversary cannot effect a change in the path prefix, he may still attempt to deny legitimate service to participating nodes.

As mentioned previously, both of our protocols are resilient to attempts by a malicious node to effect a change in the prefix of a path in which it participates. Furthermore, any attempt by such a node to deny service to legitimate nodes by dropping or modifying packets will be revealed by the misbehavior detection mechanism provided by each protocol. The node would then be immediately expelled from the path for future transmissions.

5.1.4 Invalid Response

Unless responses from the recipient are authenticated, an exit node may elect to provide an invalid response to a query, either by modifying the response from the recipient or substituting its own response. Since no other nodes in the network are able to see the query or the response, it is difficult to definitively implicate such a node as misbehaving. If the exit node can effect a change in the prefix of the path by acting in such a manner, then an adversary may be able to use the predecessor attack[17], as described above, to identify the sender. Even if the exit node cannot effect a change in the prefix of the path, it may still be able to deny the sender legitimate service by fabricating plausible responses.

As we've described, both protocols are resilient to efforts by a Byzantine actor to effect a change in the prefix of a path on which it participates. If the responses are implausible, then a sender in Protocol-1 may modify the suffix of his path in order to select a new exit node, while a sender in Protocol-2 may reissue the query using a different peer as the exit node.

Handling plausible invalid responses is more complicated. In Protocol-2 a sender may issue a query multiple times using distinct exit nodes and take a majority vote. In Protocol-1 a sender may achieve a similar result by repetitively reconstructing the path from the penultimate node. The number of times that a sender needs to issue a query in order to have a probability at most ϵ of selecting the incorrect response can be approximated using a Chernoff bound and is given by

$$n \geq \frac{1}{\left(\frac{1}{2} - f\right)^2} \ln \frac{1}{\sqrt{\epsilon}}.$$

5.1.5 Path Length Analysis

An adversary that has some knowledge about his position on a path and/or about the distribution over path lengths may be able to deduce an increased probability that his predecessor is the sender. For example, if the maximum allowed path length is ten and a malicious node participating in a path knows that the suffix of the path has length nine, then the adversary knows absolutely that the malicious node's predecessor is the sender.

Protocol-1 is susceptible to this attack. During path construction, a malicious node may refuse to forward to any node with which it is not colluding and instead report that the node is unresponsive. If the behavior of the sender is to substitute some other node for the allegedly unresponsive node, then the malicious node can continue doing this until the path is extended to a node with which it is colluding. By repeating this inductively, an adversary can ensure that every node on the suffix of the path is a under his control. By examining the distribution over the

lengths of the suffixes (which he can construct empirically), the adversary may establish an increased probability that the predecessor of the first malicious node is the sender. [1]

Protocol-2 is impervious to this attack. Because the overlay uses a fixed topology and layer-encrypted messages, it is impossible for an adversary to determine a node's position on the path of a given packet. The only case for which this does not hold is when the exit node is a malicious node. If the distribution over path lengths has sufficiently low variance, then the adversary may be able to sequester the sender to some subset of the nodes on the ring with high probability. We prevent this by explicitly using a uniform distribution over path lengths in the range $[1, n]$ where n is the number of nodes in the ring. [2] Every node on the ring is therefore equally likely to be the originator.

5.1.6 Local Eclipse

A local eclipse occurs when, in a protocol using a structured overlay with a single path, both the predecessor and the successor of a legitimate node are malicious[20]. If packets are linkable as they traverse the network, then an adversary can identify packets originated by the legitimate node since they will be seen by the successor, but they will not be linkable to any packets seen by the predecessor.

This attack is not applicable to Protocol-1 because it uses ad-hoc path creation and senders do not have predecessors. It also is not applicable to Protocol-2, despite the use of a structured overlay with a single path, because layered encryption guarantees that packets are not linkable as they pass through the network and mix with other traffic.

5.1.7 Total Eclipse

A total eclipse occurs when every node on a path except for the sender is malicious[20]. In such cases an adversary can identify the good node as the originator with high probability, and, if the protocol uses a structured overlay, possibly even certainty. We are not aware of a protocol that provides strong anonymity guarantees in the face of a total eclipse.

In Protocol-1, the probability of a total eclipse can be made small by using a sufficiently long path. Since a sender gets to select the subset of peers to form the path along which his query is routed, the probability of a total eclipse is given by f^j where j is the length of the path. Over time an adversary may be able to construct a distribution over path lengths based on the number of path suffixes he randomly eclipses. The adversary could then use the distribution to establish the probability that the successor of the first node on the suffix is the originator. This is particularly true of the version of the protocol that uses ad-hoc key exchange rather than a PKI because path construction can be hijacked. However, this is no longer a total eclipse, as it reduces to path length analysis, as described above.

In Protocol-2, a total eclipse occurs when every node on a ring except for one is malicious. Obviously in such a case the legitimate node has no anonymity. The probability of such an event effectively reduces to the probability that at least half of the nodes selected for the ring are malicious. If that were to happen, malicious nodes could falsely accuse legitimate nodes in order to remove them from the ring until only a single legitimate node remains. The probability that more than half of the nodes in a ring are *legitimate* is approximated by

$$\begin{aligned} \Pr\left[X > \frac{n}{2}\right] &\approx \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} \binom{n}{i} i^{(1-f)} (n-i)^{f} \\ &\geq 1 - e^{-2n(\frac{1}{2}-f)^2} \end{aligned}$$

where X is a random variable over the number of legitimate nodes in the ring, n is the size of the ring, and the inequality is given by the Chernoff bound. [3] We can limit the probability that more than half of the nodes in a ring are malicious to some small constant ϵ by joining only sufficiently large rings:

$$\epsilon \geq e^{-2n(\frac{1}{2}-f)^2} \Rightarrow n \geq \frac{1}{\left(\frac{1}{2}-f\right)^2} \ln \frac{1}{\sqrt{\epsilon}}.$$

A natural question to ask at this point is how certain a legitimate node can be that it is not totally eclipsed after some number of nodes have left the ring due to accusations of misbehavior. If we assume that every accusation results in the departure of one legitimate node and one malicious node, and if the size of the ring following the departures is m, then it's not hard to see that the probability that more than half of

[1] For the version of Protocol-1 that uses a PKI or pairwise key exchange, we can reduce the risk of this attack by immediately expelling both the accusing node and the accused node. We should figure out what the probability of constructing a path with at least one good node is under such a scenario. This will not help with the version Protocol-1 that performs key exchange during path construction because a malicious node can hijack the path and masquerade as the set of all of the nodes on the suffix of the path.

[2] Note that this is not equivalent to using a uniform distribution over lengths in the range $[0, n-1]$ because in that case a malicious node knows that its successor is not the originator of any packet for which the malicious node is not the exit node.

[3] This is only approximate because when we select nodes for a ring, we actually sample without replacement. This assumption is reasonable if the pool of potential nodes is much larger than the size of the ring. The true distribution is hypergeometric.

the remaining nodes are legitimate is conservatively bounded below by

$$\Pr\left[X > \frac{m}{2}\right] \geq 1 - e^{-2m(\frac{1}{2}-f)^2}.$$

That is, we can be at least as confident as we would have been had the size of the ring initially been m.

5.1.8 Denial of Service

Byzantine nodes may attempt to fill the network with so much traffic that legitimate nodes are unable to communicate. In anonymity protocols designed to obfuscate traffic as it traverses the network it can be difficult, if not impossible, to distinguish between legitimate and illegitimate traffic. This attack is more problematic for protocols that use fixed structured overlays than it is for protocols that use ad-hoc path creation simply because the nodes participating in an overlay are unable to reroute traffic to avoid congested links.

Protocol-1 is resilient to denial of service attacks. If some subset of the links in the network are congested, causing packets to be dropped, then associated nodes will be identified using the misbehavior detection mechanism and removed from any paths on which they appear. All such paths will be reconstructed from the first reliable node on the path.

Protocol-2 is also resilient to denial of service attacks. Initially one might expect that malicious nodes could simply inject enough illegitimate traffic into the ring to overwhelm participating nodes. A particularly nasty aspect of this attack is that such traffic can be injected into the ring gradually and inconspicuously since it traverses the ring until the exit node flag randomly decrypts to all zeroes. For a flag field of nontrivial size this could mean hundreds or thousands of expected ring traversals before the traffic is identified and removed.

By including a maximum sending rate for a ring, we ensure that a malicious node cannot inject illegitimate traffic indefinitely without getting caught. If he were to attempt to do so, his successor would immediately accuse him of violating the protocol once his rate of sending surpassed b_{max}, and they would both be evicted from the ring. It is interesting, however, to consider the question of what happens if a malicious node were to attempt to inject such traffic opportunistically, in order to keep his sending rate as close to the bandwidth cap as possible. In this way, his successor should never be able to send new packets, since he must keep forwarding everything he receives, which uses all of his alotted bandwidth. In theory, at this point the successor can only send by dropping a packet forwarded to him. Theoretically, this should not cause too much of a problem, since the accusation protocols are not that haretriggered, and the sending node of the dropped packet should just pick a new exit node and send again. However, if the successor wants to send a larger amount of traffic, he will need to drop many packets, which increases his likelihood of being accused and evicted from the ring.

Suppose that such a node is receiving b_{max} messages within some time frame, and wishes to send r of his own additional messages within the same window. In this case, the rate at which forwarding messages will be dropped is also equal to r. Furthermore, suppose that the protocol specifies that nodes will try to resend messages that time out some number $k-1$ times, then probe the ring for faulty nodes. In this case, the flooded sender cannot drop a particular message from the same node k times, or else he will be caught and evicted with high probability. Each packet dropped has probability $P = 1/n$ of coming from any particular node, and so the probability of dropping exactly k packets from some node n_i out of r total packets dropped is given by the binomial distribution

$$P(k, r, 1/n) = \binom{r}{k}(1/n)^k(1-1/n)^{n-k}$$

Therefore, the probability of dropping at least the requisite k from the same node n_i can be computed as a Chernoff bound (we assume $k < r/n$, which is the expected value of X in this case):

$$\begin{aligned} P(X > k) &= \sum_{i=k}^{r} P(i, r, 1/n) \\ P(X > (1-(1-\frac{k}{r/n}))\frac{r}{n}) &< e^{-r/n((1-\frac{k}{r/n}))^2/2} \end{aligned}$$

This gives us an upper bound on the probability with which we expect node n_i to be removed from the ring for dropping packets due to a DoS flood bounded above by b_{max}.

In addition to imposing a cap on the transmission rate of the ring, we can dynamically select the size of the exit node flag in order to balance the probability that it randomly decrypts to all zeroes for a legitimate packet, which we don't want to happen, against the probability that it decrypts to all zeroes for illegitimate traffic, which we do want to happen.

An adversary with sufficient network resources can still attempt a denial of service traffic simply by indiscriminately overwhelming all nodes participating in the protocol. We regard this as an orthogonal problem, as it is equivalent to the problem of preventing denial of service attacks in general. We are only interested in addressing denial of service attacks made possible by specific characteristics of our protocols.

5.1.9 Traffic Analysis

An adversary may attempt to map the path that packets take through the network by actively communicating with participating nodes in order to effect an observable change in latency. In [4], Murdoch and Danezis demonstrate how such an attack can be applied to Tor[13] if the adversary has access to information about latencies at the query recipient (or the exit node). They further claim that the techniques they present are applicable to all low-latency anonymity protocols.

Both of our protocols are susceptible to this type of attack. The path construction and query routing primitives in Protocol-1 are closely related to those in [13], so the techniques presented in [4] are directly applicable. A small nuance, however, is that in Tor, only a subset of the nodes in the network act as onion routers[13], whereas in Protocol-1 all participating nodes act as onion routers. If the number of nodes participating in Protocol-1 is large, then the adversary will require substantial resources to engage in this type of active attack, though it is certainly still possible.

Because Protocol-2 uses a structured overlay with a fixed ring topology, this attack may in fact be simpler for an adversary to carry out. If the adversary controls the server with which a participating node is communicating (or the exit node responsible for relaying the queries), then the he may attempt to increase the latency of the communication by engaging some node on the ring. If the latency of the communication does (does not) increase then the node is (is not) on the corresponding path. If the adversary identifies a node a such that engaging a results in increased latency, but engaging a's predecessor does not, then a is the sender with high probability. Using a binary search, the adversary can find this node in at most $\log_2 n$ time, where n is the number of nodes on the ring.

We note that traffic analysis of this type is predicated on the ability to make accurate measurements of latency in the network.[4] We can make such analysis much more difficult by introducing random delays to packet forwarding, as described above in the section on intersection attacks. The problem with this approach is that our protocols then become impractical for use in low-latency applications. Murdoch's and Danezis' claim notwithstanding, the question of whether it is possible to develop a low-latency anonymity protocol robust to such attacks is an open question.

5.1.10 Sampling Response

In protocols that use broadcast in order to provide the receiver's response to the sender, a malicious exit node may elect to instead send the response to some small subset of m participating nodes and then wait to see if the query is resent. In both cases an adversary can reduce the cardinality of the set of possible senders. If the query is not resent, then the sender must be among the m nodes to which the response was sent; otherwise the sender must be among the $n-m$ nodes to which the response was not sent, where n is the number of participating nodes. If the adversary identifies linkable queries then he may further reduce the number of possible senders by intersecting the corresponding sets. Eventually the adversary will be able to identify the sender with high probability.

It is possible to employ a rebroadcast scheme, whereby nodes that receive a broadcast response then rebroadcast it to some subset of nodes on the ring. This produces a cascade of broadcasts that prevents a malicious exit node from sampling using a selective response, but it requires an inordinate amount of traffic in order to ensure that every node receives the response.

Neither of protocols is vulnerable to this attack. In both cases responses are forwarded along a fixed path, so a malicious exit node cannot selectively forward the response.

5.2 Provable k-Anonymity

The anonymity provided by Protocol-1 is equal to the number of legitimate nodes participating in the protocol. There is no method by which a node may alter this anonymity, so we will not discuss it further.

The anonymity provided by a ring in Protocol-2 is exactly equal to the number of legitimate nodes participating in the ring. In our discussion on total eclipses, we show that we can limit the probability of joining a ring in which at least half of the nodes are malicious to at most ϵ by joining rings that satisfy

$$n \geq \frac{1}{\left(\frac{1}{2} - f\right)^2} \ln \frac{1}{\sqrt{\epsilon}}.$$

The same argument tells us that such a ring provides k-anonymity at least equal to $\left\lfloor \frac{n}{2} \right\rfloor + 1$ with probability at least $1 - \epsilon$ when it is constructed. If rings were static that would be sufficient.

However, as nodes accuse one another something interesting happens. If we assume that every accusation results in the expulsion of one malicious node and one legitimate node, then the fraction of malicious nodes decreases monotonically, *but so does the*

anonymity. To account for this, we must double-count the malicious nodes. The probability with that there are fewer than k legitimate nodes *after the expulsions* is given by

$$\begin{aligned}\Pr[X < k] &= \sum_{i=0}^{k-1} \binom{n}{i} i^{(1-2f)}(n-i)^{2f} \\ &< \exp\left[-\frac{(1-2f)n}{2}\left(1-\frac{k}{(1-2f)n}\right)^2\right]\end{aligned}$$

where the upper bound is derived using a Chernoff bound. Thus a user can achieve any desired amount of anonymity with high probability by joining a sufficiently large ring.

5.3 Efficiency

The proposed protocols seek to achieve k-anonymous communication, but at the cost of inducing some amount of latency into the communications channel. Protocol-1 requires all queries to travel down a "tunnel" of nodes before they can be relayed to the recipient, and all responses must be returned back up the same tunnel. Protocol-2 essentially does the same thing along an arc of the ring-shaped overlay, but with the added constraint that no participating node may send at a rate above b_{max}. This certainly induces a non-trivial amount of latency, especially considering that the nodes in the overlay in either case should not be geographically proximal to each other, so a message exchange between nodes may already require a delay on the order of seconds. However, both protocols also have the advantage that the sender may choose the length of his sending path, and so senders have the option to select a shorter path for higher-priority messages if the latency on longer paths is overly detrimental. Of course, it would not be beneficial for all nodes to do so for every message, as the varying distribution of path lengths provides some protection against statistical attacks that try to guess the probability of a malicious node being near the sender based on some computation of expected path length.

In section 6, we discuss some potential simulations and experiments we would like to perform to attempt to empirically bound the amount of latency induced by the protocols.

5.3.1 Network Overhead

Both protocols also require some amount of additional communication between nodes in order to facilitate the anonymous communication. In our analysis of this overhead, we will ignore responses from the recipient, as they can be viewed as independent communications.

Protocol-1 requires a signed receipt each time a participating node forwards a message. Since each receipt is constant in size, to send a message down a path of length l requires $O(l)$ receipts to be exchanged. Of course, since sending the communication itself requires $O(l)$ messages to be forwarded, this essentially doubles the number of messages that must be exchanged. Furthermore, there is a one-time setup phase for construction of the anonymous tunnel. Since this process iteratively sends a message down the tunnel to each node as it is added, the total number of messages required is

$$\sum_{i=i}^{l} i = \frac{l(l+1)}{2} \in O(l^2)$$

This overhead of $O(l^2)$ is only paid once, so if the sender were to reuse the established tunnel for some number $m > l$ of repeated communications, the overhead for each communication amortizes out to $O(l)$. Finally, the protocol specifies that probe messages may also be used to verify the integrity of the tunnel. However, these probe messages are to be infrequent, and actually incur the same cost as sending a query through the tunnel, so they too require a linear amount of overhead. Therefore, we may summarize that if n participants are sending through tunnels of expected length l, the total overhead over the entire overlay is $O(nl)$ in expectation.

Protocol-2 also requires a signed receipt for each message. However, this version of the protocol requires a significantly more expensive $O(n^2)$ one-time key exchange during the setup phase, which fortunately only needs to be done once, and so may be amortized out by repeated communications, especially since multiple senders may send in the same ring. Furthermore, Protocol-2 specifies that all nodes send probe messages at a constant rate, rather then just infrequently, in order to provide noise to mask senders from local observers. If we assume that an expected c probe messages are to be sent by a node for every query, and that each node uses a path of expected length l, the overhead incurred by Protocol-2 is also $O(cnl) \in O(nl)$.

5.3.2 Local Computation and Storage

Both protocols do impose a fairly heavy computation and storage requirement on participating nodes. Protocol-1 requires each node to store all public keys for his sending tunnel ($O(l)$ space), as well as a $flowid$ mapping and receipt for each communication along

each tunnel he is participating in. If we recall that the number of tunnels a node should be participating in depends linearly on $l = E[k]$, the expected path length, and additionally suppose that nodes only store their last d mappings and receipts, then each node requires an additional $O(ld)$ space (in expectation). In terms of computation, each message being sent requires l encryption operations at the sender, and each message being relayed by any node requires one layered decryption, one RSA signature, and one verification for the signed receipt. In practice, the computational complexity here should be dominated by the asymmetric operations. In the future, we hope to be able to gather empirical results as to the latency induced by these expensive operations.

Protocol-2 doesn't require $flowids$, and each node in the overlay sends along only one path, which reduces the spacial storage constraints. However, each node is responsible to remember the public keys of all other participating nodes in the overlay and the permutation of participants used to encode the ring (both of which are $O(n)$), in addition to caching some number d of receipts for messages he has forwarded in the past. The overall space requirement for each participating node is therefore $O(n + d)$. Furthermore, the same cryptographic operations are required to forward a message, and so the same latency cost will be incurred for each message sent.

6 Conclusion

The protocols we have proposed are designed to provide provable k-anonymity to communicants through the use of an anonymizing overlay. In particular, our protocols are innovative in that they provide mechanisms for the detection and removal of Byzantine entities from such an overlay, providing a strong guarantee of robustness and anonymity against adversarial attack. We have analyzed these protocols for resiliency to malicious attack, guarantees of anonymity, efficiency, and overhead. However, there is still significant work to be done:

Empirical Practicality. We'd really like to code up some simulators and then an actual client, so that we can gather some data on the actual latency and overhead induced by such a system. We've examined such concerns theoretically, but feel that real-world experiments on at least a small overlay playing the protocols is necessary to confirm the practicality of our work.

Provable Anonymity. While we are at least intuitively convinced that the proposed protocols do provide sender k-anonymity and do not leak any information regarding the sender's identity to a malicious entity, we believe so mostly because we designed the protocols to be resistant to many of the most crippling attacks we could think of. We would therefore like to examine this more formally, using an information-theoretic model of provable anonymity to see how much information a malicious entity is actually able to learn while playing the protocols.

Relaxing Assumptions. We made several assumptions while designing the protocols that we are not exactly happy about. In particular, we chose to punt on bootstrapping the protocols, our dependence on the security of IP, and the PKI in Protocol-1. We feel that some more work is needed to relax or eliminate the need for these assumptions.

Efficiency Optimizations. A few of the pieces of the protocols, as proposed, are just downright inefficient. In particular, we'd like to reexamine the $O(n^2)$ key exchange in Protocol-2 and see if we can come up with a cleaner and less costly method of sharing secret keys between pairs of participating nodes.

Open Questions. There do remain a few open questions about anonymizing overlays in general which have been touched on, but not really dealt with, by our work. In particular, it is unclear whether any anonymizing overlay protocol can truly be secure against long-term intersection attacks [3] and active traffic analysis attacks [4]. These questions continue to provide avenues of future exploration in the field of anonymous communication.

References

[1] S. Katti, J. Cohen, and D. Katabi. Information Slicing: Anonymity Using Unreliable Overlays. In *Proc. of the 4th USENIX Symp. on Networked Systems Design & Implementation*, 2007.

[2] P. Wang, P. Ning, and D. Reeves. A k-Anonymous Communication Protocol for Overlay Networks. In *Proc. of the 2nd ACM Symposium on Information, Computer and Communications Security*, 2007.

[3] O. Berthold and H. Langos. Dummy Traffic against Long Term Intersection Attacks. In *Proc. of Privacy Enhancing Technologies workshop*, 2002.

[4] S. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *Proc. of the 2005 IEEE Symp. on Security and Privacy*, 2005.

[5] J. Douceur. The Sybil Attack. In *Proc. of the 1st Int'l. Peer-to-Peer Systems Worskhop*, 2002.

[6] M. Reiter and A. Rubin. Crowds: Anonymity for Web Transactions. In *ACM Trans. on Information and System Security*, 1998.

[7] B. Levine and C. Shields. Hordes A Multicast Based Protocol for Anonymity. In *Journal of Computer Security*, 10(3), pp. 213-240, 2002.

[8] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P5: A Protocol for Scalable Anonymous Communication. In *Proc. of the 2002 IEEE Symp. on Security and Privacy*, 2002.

[9] M. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proc. of the 9^{th} ACM Conf. on Computer and Communications Security*, 2002.

[10] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proc. of the Workshop on Privacy in the Electronic Society*, 2002.

[11] S. Goel, M. Robson, M. Polte, and E. Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. In *Cornell University technical report 2003-1890*, 2003.

[12] L. Sweeney. k-anonymity: a Model for Protecting Privacy. In *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2002.

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proc. of the 13^{th} USENIX Security Symposium*, 2004.

[14] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proc. of the 2003 IEEE Symp. on Security and Privacy*, 2003.

[15] E. Androulaki, S. Geol Choi, S. Bellovin, and T. Malkin. Reputation Systems for Anonymous Networks. In *Proc. of the 8^{th} Int'l. Symp. on Privacy Enhancing Technologies*, 2008.

[16] G. Danezis and B. Wittneben. The Economics of Mass Surveillance and the Questionable Value of Anonymous Communications. In *Proc. of the 5^{th} Workshop on the Economics of Information Security*, 2006.

[17] M. Wright, M. Adler, B. Levine, and C. Shields. The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems. In *ACM Trans. on Information and System Security*, 2004.

[18] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM*, 1981.

[19] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. In *Journal of Cryptology*, Vol. 1, pp. 65-75, 1988.

[20] A. Singh, M. Castro, P. Druschel, A. Rowstron. Defending Against Eclipse Attacks on Overlay Networks. In *Proc. of the 11^{th} Workshop on ACM SIGOPS*, 2004.

[21] R. Dingledine, M. Freedman, D. Hopwood, and D Molnar. A Reputation System to Increase MIX-net Reliability. In *Proc. of Information Hiding Workshop*, pp. 126-141, 2001.

[22] R. Dingledine and P. Syverson. Reliable MIX Cascade Networks through Reputation. In *Proc. of Financial Cryptography*, 2002.

[23] R. Dingledine, M. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In *Proc. of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[24] A. Ptzmann and M. Kohntopp. Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*, pp. 1-9, 2000.

[25] L. Ahn, A. Bortz, and N. Hopper. K-anonymous message transmission. In *Proc. of the 10^{th} ACM Conf. on Computer and Communications Security*, pp. 122-130, 2003.

[26] T. Heydt-Benjamin, A. Serjantov, and B. Defend. Nonesuch: a mix network with sender unobservability. In *Proc. of the Workshop on Privacy in the Electronic Society*, pp. 1-8, 2006.

[27] A. Nambiar and M. Wright. Salsa: A Structured Approach to Large-Scale Anonymity. In *Proc. of CCS*, 2006.

[28] F. Li, B. Luo, P. Liu, and C. Chu. A Node-failure-resilient Anonymous Communication Protocol through Commutative Path Hopping. In *Proc. of the 2010 IEEE INFOCOM*, pp. 1-9, 2010.

[29] A. Pfitzmann and M. Hansen. Anonymity, unlinkability, unobservability, pseudonymity, and identity managment - a consolidated proposal for terminology. Technical Report v0.27, TU-Dresden, February 2006.