# HORNET: High-speed Onion Routing at the Network Layer

Chen Chen
chen.chen@inf.ethz.ch
CMU/ETH Zürich

Daniele E. Asoni
daniele.asoni@inf.ethz.ch
ETH Zürich

David Barrera
david.barrera@inf.ethz.ch
ETH Zürich

George Danezis
g.danezis@ucl.ac.uk
University College London

Adrian Perrig
adrian.perrig@inf.ethz.ch
ETH Zürich

## ABSTRACT

We present HORNET, a system that enables high-speed end-to-end anonymous channels by leveraging next-generation network architectures. HORNET is designed as a low-latency onion routing system that operates at the network layer thus enabling a wide range of applications. Our system uses only symmetric cryptography for data forwarding yet requires no per-flow state on intermediate routers. This design enables HORNET routers implemented on off-the-shelf hardware to process anonymous traffic at over 93 Gb/s. HORNET is also highly scalable, adding minimal processing overhead per additional anonymous channel.

## Categories and Subject Descriptors

C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: General—*Security and protection*

## General Terms

Security, Performance

## Keywords

Anonymity; onion routing; network layer

## 1. INTRODUCTION

Recent revelations about global-scale pervasive surveillance [26] programs have demonstrated that the privacy of Internet users worldwide is at risk. These revelations suggest massive amounts of private traffic, including web browsing activities, location information, and personal communications are being harvested in bulk by domestic and foreign intelligence agencies.

To protect against these threats, several anonymity protocols, tools, and architectures have been proposed. Among the most secure schemes for anonymous communications are mix networks [30, 38, 21, 22], which provide high-latency asynchronous messaging. Onion routing networks (most notably Tor [25]), offer a balance between security and performance, enabling low-latency anonymous communication suitable for typical Internet activities (e.g., web browsing, instant messaging, etc.). Tor is the system of choice for over 2 million daily users [11], but its design as an overlay network suffers from performance and scalability issues. Tor's design requires per-connection state to be maintained by intermediate nodes, limiting the total number of concurrent anonymous connections that can take place simultaneously.

The scalability and performance limitations of anonymous networks have been partially addressed by building protocols into the network layer rather than implementing them as overlays. Among these high-performing schemes are LAP [32] and Dovetail [44], which offer network-level low-latency anonymous communication on next-generation network architectures. The high performance of both schemes, however, results in significantly degraded security guarantees; endpoints have little to no protection against adversaries that are not confined to a single network location, and payload protection relies on upper layer protocols which increases complexity.

In this paper, we present HORNET (High-speed Onion Routing at the NETwork layer), a highly-scalable anonymity system that leverages next-generation Internet architecture design. HORNET offers payload protection by default, and can defend against attacks that exploit multiple network observation points. HORNET is designed to be highly efficient: it can use short paths offered by underlying network architectures, rather than the long paths due to global redirection; additionally, instead of keeping state at each relay, connection state (including, e.g., onion layer decryption keys) is carried within packet headers, allowing intermediate nodes to quickly forward traffic without per-packet state lookup.

While this paper proposes and evaluates a concrete anonymity system, a secondary goal herein is to broadly re-think the design of low-latency anonymity systems by envisioning networks where anonymous communication is offered as an in-network service to all users. For example, what performance trade-offs exist between keeping anonymous connection state at relays and carrying state in packets? If routers perform anonymity-specific tasks, how can we ensure that these operations do not impact the processing of regular network traffic, especially in adversarial circumstances? And if the network architecture should provide some support for anonymous communication, what should that support be? Throughout the paper we consider these issues in the design of our own system, and provide intuition for the requirements of alternative network-level anonymity systems.

Specifically, our contributions are the following:

- We design and implement HORNET, an anonymity system that uses source-selected paths and shared keys between endpoints and routers to support onion routing. Unlike other onion routing implementations, HORNET routers do not keep per-flow state or perform computationally expensive operations for data forwarding, allowing the system to scale.


*CCS'15,* October 12–16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813628.

- We analyze the security of HORNET, showing that it can defend against passive attacks, and certain types of active attacks. HORNET provides stronger security guarantees than existing network-level anonymity systems.
- We evaluate the performance of HORNET, showing that its anonymous data processing speed is close to that of LAP and Dovetail (up to 93.5 Gb/s on a 120 Gb/s software router). This performance is comparable with that of today's high-end commodity routers [2].

# 2. PROBLEM DEFINITION

We aim to design a network-level anonymity system to frustrate adversaries with mass surveillance capabilities. Specifically, an adversary observing traffic traversing the network should be unable to link (at large scale) pairs of communicating hosts . This property is known as relationship anonymity [41].

We define *sender anonymity* as a communication scenario where anonymity is guaranteed for the source, but the destination's location is public (e.g., web sites for The Guardian or Der Spiegel). We define *sender-receiver anonymity* as a scenario where the anonymity guarantee is extended to the destination (e.g., a hidden service that wishes to conceal its location). Sender-receiver anonymity therefore offers protection for both ends, implying sender anonymity. Depending on users' needs, HORNET can support either sender anonymity or sender-receiver anonymity.

Since our scheme operates at the network layer, network location is the only identity feature we aim to conceal. Exposure of network location or user identity at upper layers (e.g., through TCP sessions, login credentials, or browser cookies) is out of scope for this work.

## 2.1 Network Model

We consider that provisioning anonymous communication between end users is a principal task of the network infrastructure. The network's anonymity-related infrastructures, primarily routers, assist end users in establishing temporary *anonymous sessions* for anonymous data transmission.

We assume that the network layer is operated by a set of nodes. Each node cooperates with sources to establish anonymous sessions to the intended destinations, and processes anonymous traffic within the created sessions. We require that the routing state of a node allows it to determine only the next hop. In particular, the destination is only revealed to the last node and no others. This property can be satisfied by IP Segment Routing [9], Future Internet Architectures (FIAs) like NIRA [48] and SCION [51, 12], or Pathlets [28]. In practice, our abstract notion of a node could correspond to different entities depending on the architecture on which HORNET is built. For instance, in NIRA and SCION, a node corresponds to an Autonomous System (AS); in Pathlets, a node maps to a *vnode*.

**Path and certificate retrieval.** A path is the combination of routing state of all nodes between the source and the intended destination. We assume the underlying network architecture provides a mechanism for a source to obtain such a path to a given destination. Additionally, we assume that the same mechanism allows the source to fetch the public keys and certificates[1] of on-path nodes. Note that the mechanism should be privacy-preserving: the source should not reveal its network location or intent to communicate with a destination by retrieving paths, public keys, and certificates. In Section 7.1, we further discuss how to obtain required information anonymously in selected FIAs. While a general solution represents an important avenue for future work, it remains outside of our present scope.

[1]Depending on the underlying PKI scheme, the source might need to fetch a chain of certificates leading to a trust anchor to verify each node's public key.

**Public key verification.** We assume that end hosts and on-path nodes have public keys accessible and verifiable by all entities. End hosts can retrieve the public keys of other end hosts through an out-of-band channel (e.g., websites) and verify them following a scheme like HIP [39], in which the end hosts can publish hashes of their public keys as their service names. Public keys of on-path nodes are managed through a public-key infrastructure (PKI). For example, the source node can leverage Resource Public Key Infrastructure (RPKI) [16] to verify the public keys of on-path nodes.

## 2.2 Threat Model

We consider an adversary attempting to conduct mass surveillance. Specifically, the adversary collects and maintains a list of "selectors" (e.g., targets' network locations, or higher-level protocol identifiers), which help the adversary trawl intercepted traffic and extract parts of it for more extensive targeted analysis [7]. An anonymity system should prevent an adversary from leveraging bulk communication access to select traffic that belongs to the targets. Thus an adversary has to collect and analyze all traffic and cannot reliably select traffic specific to targets unless it has access to the physical links adjacent to the targets.

We consider an adversary that is able to compromise a fraction of nodes on the path between a source and a destination. For sender anonymity, the adversary can also compromise the destination. For sender-receiver anonymity, the adversary can compromise at most one of the two end hosts. By compromising a node, the adversary learns all keys and settings, observes all traffic that traverses the compromised node, and is able to control how the nodes behave including redirecting traffic, fabricating, replaying, and modifying packets.

However, we do not aim to prevent targeted de-anonymization attacks where an adversary invests a significant amount of resources on a single or a small set of victims. Like other low-latency schemes, we cannot solve targeted confirmation attacks based on the analysis of flow dynamics [45, 34, 40]. Defending against such attacks using dynamic link padding [47] would be no more difficult than in onion routing, although equally expensive. We defer the discussion and analysis of such measures to future work.

## 2.3 Desired Properties

HORNET is designed to achieve the following anonymity and security properties:

1. **Path information integrity and secrecy**. An adversary cannot modify a packet header to alter a network path without detection. The adversary should not learn forwarding information of uncompromised nodes, node's positions, or the total number of hops on a path.
2. **No packet correlation.** An adversary who can eavesdrop on multiple links in the network cannot correlate packets on those links by observing the bit patterns in the headers or payloads. This should hold regardless of whether the observed traffic corresponds to the same packet (at different points on the network), or corresponds to different packets from a single session.
3. **No session linkage.** An adversary cannot link packets from different sessions, even between the same source and destination.
4. **Payload secrecy and end-to-end integrity**. Without compromising end hosts, an adversary cannot learn any information from the data payload except for its length and timing among sequences of packets.

# 3. HORNET OVERVIEW

The basic design objectives for HORNET are *scalability* and *efficiency*. To enable Internet-scale anonymous communication, HORNET intermediate nodes must avoid keeping per-session state (e.g., cryptographic keys and routing information). Instead, session state is offloaded to end hosts, who then embed this state into packets such that each intermediate node can extract its own state as part of the packet forwarding process.

Offloading the per-session state presents two challenges. First, nodes need to prevent their offloaded state from leaking information (e.g., the session's cryptographic keys). To address this, each HORNET node maintains a local secret to encrypt the offloaded per-session state. We call this encrypted state a *Forwarding Segment* (FS). The FS allows its creating node to dynamically retrieve the embedded information (i.e., next hop, shared key, session expiration time), while hiding this information from unauthorized third parties.

The second challenge in offloading the per-session state is to combine this state (i.e., the FSes) in a packet in such a way that each node is able to retrieve its own FS, but no information is leaked about the network location of the end hosts, the path length, or a specific node's position on the path. Learning any of this information could assist in de-anonymization attacks (see Section 5.5). To address this challenge, the source constructs an *anonymous header* (AHDR) by combining multiple FSes, and prepends this header to each packet in the session. An AHDR grants each node on the path access to the FS it created, without divulging any information about the path except for a node's previous and next nodes (see Section 4.4.1).

For efficient packet processing, each HORNET node performs one Diffie-Hellman (DH) key exchange operation once per session during setup. For all data packets within the session, HORNET nodes use only symmetric cryptography to retrieve their state, process the AHDR and onion-decrypt (or encrypt) the payload. To reduce setup delay, HORNET uses only two setup packets within a single round trip between the source and the destination. Therefore, session setup only incurs $O(n)$ propagation delay in comparison to $O(n^2)$ by the telescopic setup method used in Tor (where $n$ is the number of anonymity nodes traversed on the path). While for Tor the default value of $n$ is 3, for HORNET $n$ might be as large as 14 (4.1 in the average case, and less or equal to 7 in over 99% of cases [6]), which emphasizes the need to optimize setup propagation delay.

## 3.1 Sender Anonymity

Anonymous sessions between a source and a destination require the source to establish state between itself and every node on the path. The state will be carried in subsequent data packets, enabling intermediate nodes to retrieve their corresponding state and forward the packet to the next hop. We now describe how the state is collected without compromising the sender's anonymity, and how this state is used to forward data packets.

**Setup phase.** To establish an anonymous session between a source $S$ and a public destination $D$, $S$ uses a single round of Sphinx [22], a provably secure mix protocol (an overview of Sphinx is given in Section 4.3.1). This round consists of two Sphinx packets (one for the forward path and one for the backward path) each of which will anonymously establish shared symmetric keys between $S$ and every node on that path. For HORNET, we extend the Sphinx protocol to additionally anonymously collect the forwarding segments (FSes) for each node. Our modified Sphinx protocol protects the secrecy and integrity of these FSes, and does not reveal topology information to any node on the path. We note that using Sphinx also for data forwarding would result in low throughput due to prohibitively expensive per-hop asymmetric cryptographic operations. Therefore, we use Sphinx only for session setup packets, which are amortized over the subsequent data transmission packets. We explain the details of the setup phase in Section 4.3.

**Data transmission phase.** Having collected the FSes, the source is now able to construct a forward AHDR and a backward AHDR for the forward and backward paths, respectively. AHDRs carry the FSes which contain all state necessary for nodes to process and forward packets to the next hop. When sending a data packet, the source onion-encrypts the data payload using the session's shared symmetric keys, and prepends the AHDR. Each node then retrieves its FS from the AHDR, onion-decrypts the packet and forwards it to the next hop, until it reaches the destination. The destination uses the backward AHDR (received in the first data packet[2]) to send data back to $S$, with the only difference being that the payload is encrypted (rather than decrypted) at each hop. We present the details of the data transmission phase in Section 4.4.

## 3.2 Sender-Receiver Anonymity

Sender-receiver anonymity, where neither $S$ nor $D$ knows the other's location (e.g., a hidden service), presents a new challenge: since $S$ does not know $D$'s location (and vice versa), $S$ cannot retrieve a path to $D$, precluding the establishment of state between $S$ and nodes on the path to $D$ as described in Section 3.1.

A common approach to this problem (as adopted by Tor[3], LAP, and Dovetail) is to use a public *rendezvous point* (RP) to forward traffic between $S$ and $D$ without knowing either $S$ or $D$. This solution would also work for HORNET, but would require RPs to maintain per-session state between sources and destinations. For instance, when receiving a packet from $S$, an RP needs the state to determine how to send the packet to $D$. Maintaining per-session state on RPs increases complexity, bounds the number of receivers, and introduces a state exhaustion denial-of-service attack vector.

**Nested AHDRs.** Our proposal for sender-receiver anonymity requires no state to be kept at the RP by nesting the necessary state for RPs to forward a packet within the packet's header: a forward AHDR from $S$ to a RP will include the AHDR from the RP to $D$; a backward AHDR from $D$ to a RP will include the AHDR from the RP back to $S$.

Briefly, to establish a HORNET session between $S$ and $D$ keeping both parties hidden from each other, $D$ selects a public rendezvous point $R$ and completes a HORNET session setup between $D$ and $R$. $D$ publishes $\text{AHDR}_{R \to D}$ to a public directory. Note that this AHDR leaks no information about $D$'s location and can only be used to send data to $D$ through $R$ within a specific time window.

When $S$ wants to send traffic to $D$, $S$ retrieves (from a public directory) $\text{AHDR}_{R \to D}$. $S$ then establishes a HORNET session between $S$ and $R$ and constructs a nested AHDR with $\text{AHDR}_{R \to D}$ inside $\text{AHDR}_{S \to R}$. Thus, when $R$ receives a packet from $S$, $R$ can retrieve $\text{AHDR}_{R \to D}$ from $\text{AHDR}_{S \to R}$ and forward the packet to $D$. $S$ also includes $\text{AHDR}_{R \to S}$ in the data payload of the first data packet to $D$, allowing $D$ to create a return path to $S$.

One of the advantages of our scheme is that any node on the network can serve as a rendezvous point. In fact, multiple points can

[2]If the first packet is lost the source can simply resend the backward AHDR using a new data packet (see Section 4.4).

[3]Tor additionally uses an introduction point, which enables $S$ to negotiate a rendezvous point with $D$. This design provides additional scalability and attack resistance [25], but increases the delay of setting up a session. HORNET's design favors simplicity and performance, but nothing fundamentally prevents HORNET from using Tor's approach.

be selected and advertised, allowing the source to pick the RP closest to it. Moreover, once a HORNET session has been established, $S$ and $D$ can negotiate a better (closer) RP (e.g., using private set intersection [27]). A disadvantage of the nested AHDR technique is that it doubles the size of the header.

For space reasons, the formal protocol details and evaluation sections focus on sender anonymity only. Details of sender-receiver anonymity can be found in the full paper [19].

## 3.3 Packet Structure

HORNET uses two types of packets: *setup packets* and *data packets* (see Figure 1). Both types of packets begin with a common header (CHDR) which describes the packet type, the length of the longest path that the session supports, and a type-specific field. For session setup packets, the type-specific field contains a value EXP which indicates the intended expiration time of the session. For data packets, the specific value is a random nonce generated by the sender used by intermediate nodes to process the data packet.

Session setup packets include a nested Sphinx packet and an FS payload. Data packets carry an AHDR and an onion-encrypted data payload. We explain each field in detail in Section 4.

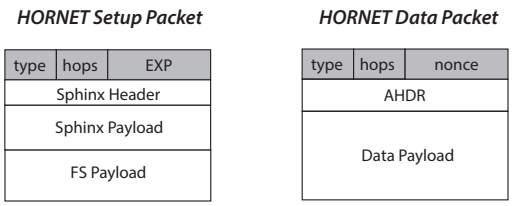



**Figure 1: HORNET packet formats. For both setup packet and data packet, the shaded fields represent the common header (CHDR).**

# 4. FORMAL PROTOCOL DESCRIPTION

We now describe the details of our protocol, focusing on sender anonymity. We begin with notation (Section 4.1) and initialization requirements (Section 4.2). We then describe the establishment of anonymous communication sessions (Section 4.3) and data transmission (Section 4.4).

## 4.1 Notation

Let $k$ be the security parameter used in the protocol. For evaluation purposes we consider $k = 128$. $\mathcal{G}$ is a prime order cyclic group of order $q$ ($q \sim 2^{2k}$), which satisfies the Decisional Diffie-Hellman Assumption. $\mathcal{G}^*$ is the set of non-identity elements in $\mathcal{G}$ and $g$ is a generator of $\mathcal{G}$. Throughout this section we use the multiplicative notation for $\mathcal{G}$.

Let $r$ be the maximum length of a path, i.e., the maximum number of nodes on a path, including the destination. We denote the length of an FS as $|FS|$ and the size of an AHDR block, containing an FS and a MAC of size $k$, as $c = |FS| + k$.

HORNET uses the following cryptographic primitives:

- $\mathsf{MAC} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$: Message Authentication Code (MAC) function.
- $\mathsf{PRG0}, \mathsf{PRG1}, \mathsf{PRG2} : \{0,1\}^k \to \{0,1\}^{rc}$: Three cryptographic pseudo-random generators.
- $\mathsf{PRP} : \{0,1\}^k \times \{0,1\}^a \to \{0,1\}^a$: A pseudo-random permutation, implementable as a block cipher. The value of $a$ will be clear from the context.

| Term | Definition |
|---|---|
| $k$ | Security parameter (length of keys and MACs). $k = 128$ bits (16 B). |
| $\|FS\|$ | Length of a forwarding segment (FS). $\|FS\| = 256$ bits (32 B). |
| $c$ | Length of a typical block made of an FS and a MAC. $c = \|FS\| + k = 384$ bits (48 B) . |
| $r$ | Maximum path length, including the destination. From our evaluation, $r = 7$. |
| $S, D$ | Source and destination. |
| $p^f, p^b$ | The forward path (from S to D) and the backward path (from D to S). |
| $l^f, l^b$ | Lengths of the forward and backward path ($l$, when it is clear from the context to which path it refers). From our evaluation, $1 \le l \le 7$. |
| $n_i^f, n_j^b$ | The $i$-th node on the forward path and the $j$-th node on the backward path, with $0 \le i < l^f$ and $0 \le j < l^b$. |
| $g^{x_n}, x_n$ | Public/private key pair of node $n$. |
| $s_i^f$ | Secret key established between S and node $n_i^f$. |
| $R$ | Routing information, which allows a node to forward a packet to the next hop. |
| CHDR | Common header. First three fields of both setup packets and data packets (see Figure 1). |
| SHDR, SP | Sphinx header and payload. |
| $P$ | FS payload, used to collect the FSes during the setup phase. |
| AHDR | Anonymous header, used for every data packet. It allows each node on the path to retrieve its FS. |
| $O$ | Onion payload, containing the data payload of data packets. |
| EXP | Expiration time, included in each FS. |

**Table 1: Protocol notation and typical values (where applicable).**

- $\mathsf{ENC} : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{mk} \to \{0,1\}^{mk}$: Encryption function, with the second parameter being the Initialization Vector (IV) (e.g., stream cipher in CBC mode). $m$ is a positive integer denoting the number of encrypted blocks.
- $\mathsf{DEC} : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^{mk} \to \{0,1\}^{mk}$: Decryption function, inverse of ENC.
- $h_{\mathsf{op}} : \mathcal{G}^* \to \{0,1\}^k$: a family of hash functions used to key op, with $\mathsf{op} \in \{\mathsf{MAC}, \mathsf{PRG0}, \mathsf{PRG1}, \mathsf{PRP}, \mathsf{ENC}, \mathsf{DEC}\}$.

We denote by $\text{RAND}(a)$ a function that generates a new uniformly random string of length $a$.

Furthermore, we define the notation for bit strings. $0^a$ stands for a string of zeros of length $a$. $|\sigma|$ is the length of the bit string $\sigma$. $\sigma_{[a...b]}$ represents a substring of $\sigma$ from bit $a$ to bit $b$, with subindex $a$ starting from 0; $\sigma_{[a...end]}$ indicates the substring of $\sigma$ from bit $a$ till the end. $\varepsilon$ is the empty string. $\sigma \,||\, \sigma'$ is the concatenation of string $\sigma$ and string $\sigma'$. We summarize protocol notation and typical values for specific parameters in Table 1.

In the following protocol description, we consider a source $S$ communicating with a destination $D$ using forward path $p^f$ traversing nodes $n_0^f, n_1^f, \ldots, n_{l^f-1}^f$ and backward path $p^b$ traversing nodes $n_0^b, n_1^b, \ldots, n_{l^b-1}^b$, with $l^f, l^b \le r$, where $n_0^f$ and $n_{l^b-1}^b$ are the nodes closest to the source. Without loss of generality, we let the last node on the forward path $n_{l^f-1}^f = D$ and refer to the destination by these two notations interchangeably. In general we use $dir \in \{f, b\}$ as superscripts to distinguish between notation referring to the forward and backward path, respectively. Finally, to avoid redundancy, we use $\{sym_i^{dir}\}$ to denote $\{sym_i^{dir} | 0 \le i \le l^{dir} - 1\}$, where $sym$ can be any symbol.

## 4.2 Initialization

Suppose that a source $S$ wishes to establish an anonymous session with a public destination $D$. First, $S$ anonymously obtains (from the underlying network) paths in both directions: a forward path $p^f = \{R_0^f, R_1^f, \cdots, R_{l^f-1}^f\}$ from $S$ to $D$ and a backward path $p^b = \{R_0^b, R_1^b, \cdots, R_{l^b-1}^f\}$ from $D$ to $S$. $R_i^{dir}$ denotes the routing information needed by the node $n_i^{dir}$ to forward a packet. $S$ also anonymously retrieves and verifies a set of public keys $g^{x_{n_i^{dir}}}$ for the node $n_i^{dir}$ on path $p^{dir}$ (see Section 2.1). Note that $g^{x_D}$ is also included in the above set (as $n_{l^f-1}^f = D$). Finally, $S$ generates a random DH public/private key pair for the session: $x_S$ and $g^{x_S}$. The per-session public key $g^{x_S}$ is used by the source to create shared symmetric keys with nodes on the paths later in the setup phase. $S$ locally stores $\left\{(x_S, g^{x_S}), \left\{g^{x_{n_i^{dir}}}\right\}, p^{dir}\right\}$, and uses these values for the setup phase.

## 4.3 Setup Phase

As discussed in Section 3, in the setup phase, HORNET uses two Sphinx packets, which we denote by P❶ and P❷, to traverse all nodes on both forward and backward paths and establish per-session state with every intermediate node, without revealing $S$'s network location. For $S$ to collect the generated per-session state from each node, both Sphinx packets contain an empty FS payload into which each intermediate node can insert its FS, but is not able to learn anything about, or modify, previously inserted FSes.

### *4.3.1 Sphinx Overview*

Sphinx [22] is a provably-secure mix protocol. Each Sphinx packet allows a source node to establish a set of symmetric keys, one for each node on the path through which packets are routed. These keys enable each node to check the header's integrity, onion-decrypt the data payload, and retrieve the information to route the packet. Processing Sphinx packets involves expensive asymmetric cryptographic operations, thus Sphinx alone is not suitable to support high-speed anonymous communication.

**Sphinx packets.** A Sphinx packet is composed of a Sphinx header SHDR and a Sphinx payload SP. The SHDR contains a group element ${y_i}^{dir}$ that is re-randomized at each hop. Each ${y_i}^{dir}$ is used as $S$'s ephemeral public key in a DH key exchange with node $n_i^{dir}$. From this DH exchange, node $n_i^{dir}$ derives a shared symmetric key $s_i^{dir}$, which it uses to process the rest of the SHDR and mutate ${y_i}^{dir}$. The rest of the SHDR is an onion-encrypted data structure, with each layer containing routing information and a MAC. The routing information indicates to which node the packet should be forwarded to next, and the MAC allows to check the header's integrity at the current node. The Sphinx payload SP allows end hosts to send confidential content to each other. Each intermediate node processes SP by using a pseudo-random permutation.

**Sphinx core functions.** We abstract the Sphinx protocol into the following six functions:

- GEN_SPHX_HDR. The source uses this function to generate two Sphinx headers, $\text{SHDR}^f$ and $\text{SHDR}^b$, for the forward and backward path, respectively. It also outputs the symmetric keys $\{s_i^{dir}\}$, each established with the corresponding node's public key $g^{x_{n_i^{dir}}}$.
- GEN_SPHX_PL_SEND. The function allows the source to generate an onion-encrypted payload $\text{SP}^f$ encapsulating confidential data to send to the destination.
- UNWRAP_SPHX_PL_SEND. The function removes the last encryption layer added by GEN_SPHX_PL_SEND, and allows the destination to decrypt the $\text{SP}^f$.
- GEN_SPHX_PL_RECV. The function enables the destination to cryptographically wrap a data payload into $\text{SP}^b$ before sending it to the source.
- UNWRAP_SPHX_PL_RECV. The function allows the source to recover the plaintext of the payload that the destination sent.
- PROC_SPHX_PKT. Intermediate nodes use this function to process a Sphinx packet, and establish symmetric keys shared with the source. The function takes as inputs the packet $(\text{SHDR}, \text{SP})$, and the node's DH public key $g^{x_{n_i^{dir}}}$. The function outputs the processed Sphinx packet $(\text{SHDR}', \text{SP}')$ and the established symmetric key $s_i^{dir}$.

### *4.3.2 Forwarding Segment*

We extend Sphinx to allow each node to create a Forwarding Segment (FS) and add it to a data structure we name FS payload (see below). An FS contains a node's per-session state, which consists of a secret key $s$ shared with the source, a routing segment $R$, and the session's expiration time EXP. To protect these contents, the FS is encrypted with a PRP keyed by a secret value $SV$ known only by the node that creates the FS. A node seals and unseals its state using two opposite functions: FS_CREATE and FS_OPEN. They are defined as follows:

$$\begin{aligned} FS &= \text{FS\_CREATE}(SV, s, R, \text{EXP}) = \\ &= \mathsf{PRP}(h_{\mathsf{PRP}}(SV); \{s \,\|\, R \,\|\, \text{EXP}\}) \end{aligned} \tag{1}$$

$$\begin{aligned} \{s \,\|\, R \,\|\, \text{EXP}\} &= \text{FS\_OPEN}(SV, FS) \\ &= \mathsf{PRP}^{-1}(h_{\mathsf{PRP}}(SV); FS) \end{aligned} \tag{2}$$

### *4.3.3 FS Payload*

At the end of each HORNET setup packet is a data structure we call FS payload (see Figure 1). The FS payload is an onion-encrypted construction that allows intermediate nodes to add their FSes as onion-layers.

Processing the FS payload leaks no information about the path's length or about an intermediate node's position on the path. All FS payloads are padded to a fixed length, which is kept constant by dropping the right number of trailing bits of the FS payload before an FS is added to the front. Moreover, new FSes are always added to the beginning of the FS payload, eliminating the need for intermediate nodes to know their positions in order to process FS payloads.

An FS payload also provides both secrecy and integrity for the FSes it contains. Each node re-encrypts the FS payload after inserting a new FS and computes a MAC over the resulting structure. Only the source, with symmetric keys shared with each node on a path, can retrieve all the FSes from the FS payload and verify their integrity.

**Functions.** There are three core functions for the FS payload: INIT_FS_PAYLOAD, ADD_FS, and RETRIEVE_FSES.

INIT_FS_PAYLOAD. A node initializes an FS payload by using a pseudo-random generator keyed with a symmetric key $s$ to generate $rc$ random bits:

$$P = \mathsf{PRG1}(h_{\mathsf{PRG1}}(s)) \tag{3}$$

where $c = |FS| + k$ is the size of a basic block of the FS payload (consisting of an FS and a MAC).

ADD_FS. Each intermediate node uses ADD_FS to insert its FS into the payload, as shown in Algorithm 1. First, the trailing $c$ bits

**Algorithm 1** Add FS into FS payload.

1: **procedure** ADD_FS
   Input: $s$, $FS$, $P_{in}$
   Output: $P_{out}$
2: $P_{tmp} \leftarrow \left\{ FS \,\|\, P_{in[0..(r-1)c-1]} \right\} \oplus \mathsf{PRG0}(h_{\mathsf{PRG0}}(s))_{[k..end]}$
3: $\alpha \leftarrow \mathsf{MAC}(h_{\mathsf{MAC}}(s); P_{tmp})$
4: $P_{out} \leftarrow \alpha \,\|\, P_{tmp}$
5: **end procedure**

**Algorithm 2** Retrieve FSes from FS payload.

1: **procedure** RETRIEVE_FSES
   Input: $P$, $s$, $\{s_i\}$
   Output: $\{FS_i\}$
2: $P_{init} \leftarrow$ INIT_FS_PAYLOAD$(s)$
3: $\psi \leftarrow P_{init[(r-l)c..rc-1]} \oplus \mathsf{PRG0}(h_{\mathsf{PRG0}}(s_0))_{[(r-l+1)c..end]} \,\|\, 0^c \oplus \mathsf{PRG0}(h_{\mathsf{PRG0}}(s_1))_{[(r-l+2)c..end]} \,\|\, 0^{2c} \cdots \oplus \mathsf{PRG0}(h_{\mathsf{PRG0}}(s_{l-2}))_{[(r-1)c..end]} \,\|\, 0^{(l-1)c}$
4: $P_{full} = P \,\|\, \psi$
5: **for** $i \leftarrow (l-1), \ldots, 0$ **do**
6: **check** $P_{full[0..k-1]} = \mathsf{MAC}(h_{\mathsf{MAC}}(s_i); P_{full[k..rc-1]})$
7: $P_{full} \leftarrow P_{full} \oplus (\mathsf{PRG0}(h_{\mathsf{PRG0}}(s_i)) \,\|\, 0^{(i+1)c})$
8: $FS_i \leftarrow P_{full[k..c-1]}$
9: $P_{full} \leftarrow P_{full[c..end]}$
10: **end for**
11: **end procedure**

of the current FS payload, which are padding bits containing no information about previously added FSes, are dropped, and then the FS is prepended to the shortened FS payload. The result is encrypted using a stream cipher (Line 2) and MACed (Line 4). Note that no node-position information is required in ADD_FS, and verifying that the length of the FS payload remains unchanged is straightforward.

RETRIEVE_FSES. The source uses this function to recover all FSes $\{FS_i\}$ inserted into an FS payload $P$. RETRIEVE_FSES starts by recomputing the discarded trailing bits (Line 3) and obtaining a complete payload $P_{full}$. Thus, intuitively, this full payload is what would remain if no nodes dropped any bits before inserting a new FS. Afterwards, the source retrieves the FSes from $P_{full}$ in the reverse order in which they were added by ADD_FS (see lines 6 and 8).

### 4.3.4 Setup Phase Protocol Description

**Source processing.** With the input

$$I = \left\{ \left(x_S, g^{x_S}\right), \left\{ g^{x_{n_i^{dir}}} \right\}, p^{dir} \right\}$$

the source node $S$ bootstraps a session setup in 5 steps:

1. $S$ selects the intended expiration time EXP for the session and specifies it in the common header CHDR (see Section 3.3).[4]
2. $S$ generates the send and the reply Sphinx headers by:
$$\{\text{SHDR}^f, \text{SHDR}^b\} = \text{GEN\_SPHX\_HDR}(I, \text{CHDR}) \quad (4)$$
The common header CHDR (see Figure 1) is passed to the function to extend the per-hop integrity protection of Sphinx over it. GEN_SPHX_HDR also produces the symmetric keys shared with each node on both paths $\{s_i^{dir}\}$.
3. In order to enable the destination $D$ to reply, $S$ places the reply Sphinx header $\text{SHDR}^b$ into the Sphinx payload:
$$\text{SP}^f = \text{GEN\_SPHX\_PL\_SEND}(\{s_i^f\}, \text{SHDR}^b) \quad (5)$$
4. $S$ creates an initial FS payload $P^f = \text{INIT\_FS\_PAYLOAD}(x_S)$.
5. $S$ composes $\mathsf{P❶} = \{\text{CHDR} \,\|\, \text{SHDR}^f \,\|\, \text{SP}^f \,\|\, P^f\}$ and sends it to the first node on the forward path $n_0^f$.

**Intermediate node processing.** An intermediate node $n_i^f$ receiving a packet $\mathsf{P❶} = \{\text{CHDR} \,\|\, \text{SHDR}^f \,\|\, \text{SP}^f \,\|\, P^f\}$ processes it as follows:

1. $n_i^f$ first processes $\text{SHDR}^f$ and $\text{SP}^f$ in $\mathsf{P❶}$ according to the Sphinx protocol (using PROC_SPHX_PKT). As a result $n_i^f$ obtains the established symmetric key $s_i^f$ shared with $S$, the processed header and payload $(\text{SHDR}^{f'}, \text{SP}^{f'})$ as well as the routing information $R_i^f$. During this processing the integrity of the CHDR is verified.
2. $n_i^f$ obtains EXP from CHDR and checks that EXP is not expired. $n_i^f$ also verifies that $R_i^f$ is valid.
3. $n_i^f$ generates its forwarding segment $FS_i^f$ by using its local symmetric key $SV_i^f$ to encrypt $s_i^f$, $R_i^f$, and EXP (see Equation 1):
$$FS_i^f = \text{FS\_CREATE}(SV_i^f, s_i^f, R_i^f, \text{EXP}) \quad (6)$$
4. $n_i^f$ adds its $FS_i^f$ into the FS payload $P^f$.
$$P^{f'} = \text{ADD\_FS}(s_i^f, FS_i^f, P^f) \quad (7)$$
5. Finally node $n_i^f$ assembles the processed packet $\mathsf{P❶} = \{\text{CHDR} \,\|\, \text{SHDR}^{f'} \,\|\, \text{SP}^{f'} \,\|\, P^{f'}\}$ and routes it to the next node according to the routing information $R_i^f$.

**Destination processing.** As the last node on the forward path, $D$ processes $\mathsf{P❶}$ in the same way as the previous nodes. It first processes the Sphinx packet in $\mathsf{P❶}$ and derives a symmetric key $s_D$ shared with $S$, and then it encrypts per-session state, including $s_D$, into $FS_D$, and inserts $FS_D$ into the FS payload.

After these operations, however, $D$ moves on to create the second setup packet $\mathsf{P❷}$ as follows:

1. $D$ retrieves the Sphinx reply header using the symmetric key $s_D$:
$$\text{SHDR}^b = \text{UNWRAP\_SPHX\_PL\_SEND}(s_D, \text{SP}^f) \quad (8)$$
2. $D$ places the FS payload $\text{P}^f$ of $\mathsf{P❶}$ into the Sphinx payload $\text{SP}^b$ of $\mathsf{P❷}$ (this will allow $S$ to get the FSes $\{FS_i^f\}$):
$$\text{SP}^b = \text{GEN\_SPHX\_PL\_RECV}(s_D, P^f) \quad (9)$$
Note that since $D$ has no knowledge about the keys $\{s_i^f\}$ except for $s_D$, $D$ learns nothing about the other FSes in the FS payload.

[4] EXP must not become an identifier that allows matching packets of the same flow across multiple links. Since EXP does not change during setup packet forwarding, a coarser granularity (e.g., 10s) is desirable. In addition, the duration of the session should also have only a restricted set of possible values (e.g., 10s, 30s, 1min, 10min) to avoid matching packets within long sessions. For long-lived connections, the source can create a new session in the background before expiration of the previous one to avoid additional latency.

3. $D$ creates a new FS payload $P^b = \text{INIT\_FS\_PAYLOAD}(s_D)$ to collect the FSes along the backward path.
4. $D$ composes $\mathsf{P}❷ = \{\text{CHDR} \,\|\, \text{SHDR}^b \,\|\, \text{SP}^b \,\|\, P^b\}$ and sends it to the first node on the backward path, $n_0^b$.

The nodes on the backward path process $\mathsf{P}❷$ in the exact same way nodes on the forward path processed $\mathsf{P}❶$. Finally $\mathsf{P}❷$ reaches the source $S$ with FSes $\{FS_i^b\}$ added to the FS payload.

**Post-setup processing.** Once $S$ receives $\mathsf{P}❷$ it extracts all FSes, i.e., $\{FS_i^f\}$ and $\{FS_i^b\}$, as follows:

1. $S$ recovers the FS payload for the forward path $P^f$ from $\text{SP}^b$:

$$P^f = \text{UNWRAP\_SPHX\_PL\_RECV}(\{s_i^b\}, \text{SP}^b) \quad (10)$$

2. $S$ retrieves the FSes for the nodes on the forward path $\{FS_i^f\}$:

$$\{FS_i^f\} = \text{RETRIEVE\_FSES}(\{s_i^f\}, P^f) \quad (11)$$

3. $S$ directly extracts from $P^b$the FSes for the nodes on the backward path $\{FS_i^b\}$:

$$\{FS_i^b\} = \text{RETRIEVE\_FSES}(\{s_i^b\}, P^b) \quad (12)$$

With the FSes for all nodes on both paths, $\{FS_i^f\}$ and $\{FS_i^b\}$, $S$ is ready to start the data transmission phase.

## 4.4 Data Transmission Phase

Each HORNET data packet contains an anonymous header AHDR and an onion-encrypted payload $O$ as shown in Figure 1. Figure 2 illustrates the details of an AHDR. The AHDR allows each intermediate node along the path to retrieve its per-session state in the form of an FS and process the onion-encrypted data payload. All processing of data packets in HORNET only involves symmetric-key cryptography, therefore supporting fast packet processing.

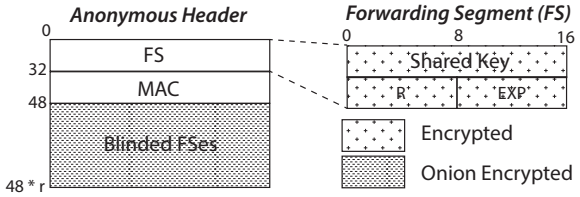



**Figure 2: Format of a HORNET anonymous header with details of a forwarding segment (FS).**

At the beginning of the data transmission phase, $S$ creates two AHDRs, one for the forward path ($\text{AHDR}^f$) and one for the backward path ($\text{AHDR}^b$), by using FSes collected during the setup phase. $\text{AHDR}^f$ enables $S$ to send data payloads to $D$. To enable $D$ to transmit data payloads back, $S$ sends $\text{AHDR}^b$ as payload in the first data packet. If this packet is lost, the source would notice from the fact that no reply is seen from the destination. If this happens the source simply resends the backward AHDR using a new data packet.

### 4.4.1 Anonymous Header

Like an FS payload, an AHDR is an onion-encrypted data structure that contains FSes. It also offers the same guarantees, i.e., secrecy and integrity, for the individual FSes it contains, for their number and for their order. Its functionalities, on the other hand, are the inverse: while the FS payload allows the source to collect the FSes added by intermediate nodes, the AHDR enables the source to re-distribute the FSes back to the nodes for each transmitted data packet.

**Algorithm 3** Process an AHDR.

**procedure** PROC_AHDR
Input: $SV$, AHDR
Output: $s$, $R$, $\text{AHDR}'$
  $\{FS \,\|\, \gamma \,\|\, \beta\} \leftarrow \text{AHDR}$
  $\{s \,\|\, R \,\|\, \text{EXP}\} \leftarrow \text{FS\_OPEN}(SV, FS)$
  **check** $\gamma = \mathsf{MAC}(h_{\mathsf{MAC}}(s); FS \,\|\, \beta)$
  **check** $t_{curr} < \text{EXP}$
  $\text{AHDR}' \leftarrow \{\beta \,\|\, 0^c\} \oplus \mathsf{PRG2}(h_{\mathsf{PRG2}}(s))$
**end procedure**

**Algorithm 4** Anonymous header construction.

**procedure** CREATE_AHDR
Input: $\{s_i\}$, $\{FS_i\}$
Output: $(FS_0, \gamma_0, \beta_0)$
  $\phi_0 \leftarrow \varepsilon$
  **for** $i \leftarrow 0, \ldots, l-2$ **do**
    $\phi_{i+1} \leftarrow (\phi_i \,\|\, 0^c) \oplus \left\{\mathsf{PRG2}(h_{\mathsf{PRG2}}(s_i))_{[(r-1-i)c..end]}\right\}$
  **end for**
  $\beta_{l-1} \leftarrow \text{RAND}\,((r-l)c) \,\|\, \phi_{l-1}$
  $\gamma_{l-1} \leftarrow \mathsf{MAC}(h_{\mathsf{MAC}}(s_{l-1}); FS_{l-1} \,\|\, \beta_{l-1})$
  **for** $i \leftarrow (l-2), \ldots, 0$ **do**
    $\beta_i \leftarrow \left\{FS_{i+1} \,\|\, \gamma_{i+1} \,\|\, \beta_{i+1}{}_{[0..(r-2)c-1]}\right\} \oplus \mathsf{PRG2}(h_{\mathsf{PRG2}}(s_i))_{[0..(r-1)c-1]}$
    $\gamma_i \leftarrow \mathsf{MAC}(h_{\mathsf{MAC}}(s_i); FS_i \,\|\, \beta_i)$
  **end for**
**end procedure**

**Functions.** The life cycle of AHDRs consists of two functions: the header construction (CREATE_AHDR) and the header processing (PROC_AHDR). We begin with the description of PROC_AHDR since it is simpler, and its helps understand the construction of CREATE_AHDR.PROC_AHDR allows each intermediate node to verify the integrity of an incoming AHDR, and to check that the corresponding session has not expired. PROC_AHDR also retrieves the key $s$ shared with the source, as well as the routing information $R$, from the FS of the node invoking the function. Finally, PROC_AHDR also returns the processed header $\text{AHDR}'$, which will be used by the next hop. The details of this function can be seen in Algorithm 3.

Our AHDR construction resembles the Sphinx packet header construction [22]. For each path (forward and backward), CREATE_AHDR enables $S$ to create an AHDR given the keys $\{s_i\}$ shared with each node on that path, and given the forwarding segments $\{FS_i\}$ of those nodes. All these keys and FSes are obtained during the setup phase (see Section 4.3). The details are shown in Algorithm 4. In essence, CREATE_AHDR is equivalent to a series of PROC_AHDR iterations performed in reverse. Initially, the paddings $\phi$ are computed, each of which is the leftmost part of an AHDR that results from the successive encryptions of the zero-paddings added in PROC_AHDR ($\phi_0$ is the empty string since no padding has been added yet). Once the last padding is computed (the one for the AHDR received by the last hop, $\phi_{l-1}$), the operations in PROC_AHDR are reversed, obtaining at each step the AHDRs as will be received by the nodes, from the last to the first. This also allows the computation of the per-hop MACs.

### 4.4.2 Onion Payload

HORNET data payloads are protected by onion encryption. To send a data payload to the destination, the source adds a sequence of encryption layers on top of the data payload, one for each node on the forward path (including the destination). As the packet is forwarded, each node removes one layer of encryption, until the destination removes the last layer and obtains the original plaintext.

To send a data payload back to the source, the destination adds only one layer of encryption with its symmetric key shared with the source. As the packet is forwarded, each node on the backward path re-encrypts the payload until it reaches the source. With all the symmetric keys shared with nodes on the backward path, the source is capable of removing all encryption layers, thus obtaining the original data payload sent by the destination.

**Functions.** Processing onion payloads requires the following two functions: ADD_LAYER and REMOVE_LAYER.

ADD_LAYER. The function's full form is:

$$\{O', IV'\} = \text{ADD\_LAYER}(s, IV, O) \tag{13}$$

Given a symmetric key $s$, an initial vector $IV$, and an input onion payload $O$, ADD_LAYER performs two tasks. First, ADD_LAYER encrypts $O$ with $s$ and $IV$:

$$O' = \mathsf{ENC}(h_{\mathsf{ENC}}(s); IV; O) \tag{14}$$

Then, to avoid making the IV an identifier across different links, ADD_LAYER mutates the $IV$ for the next node:

$$IV' = \mathsf{PRP}(h_{\mathsf{PRP}}(s); IV) \tag{15}$$

REMOVE_LAYER. The function is the inverse of ADD_LAYER, decrypting the onion payload at each step, and mutating the $IV$ using the inverse permutation $\mathsf{PRP}^{-1}$ keyed with $h_{\mathsf{PRP}}(s)$. Its full form is the following:

$$\{O', IV'\} = \text{REMOVE\_LAYER}(s, IV, O) \tag{16}$$

### 4.4.3 Initializing Data Transmission

To start the data transmission session, $S$ generates $\text{AHDR}^f$ and $\text{AHDR}^b$ as follows:

$$\text{AHDR}^f = \text{CREATE\_AHDR}(\{s_i^f\}, \{FS_i^f\}) \tag{17}$$

$$\text{AHDR}^b = \text{CREATE\_AHDR}(\{s_i^b\}, \{FS_i^b\}) \tag{18}$$

$S$ then sends $\text{AHDR}^b$ to $D$ as payload of the first data packet (which uses $\text{AHDR}^f$), as specified in the following section.

### 4.4.4 Data Transmission Protocol Description

**Source processing.** With $\text{AHDR}^f$, $S$ can send a data payload $P$ with the following steps:

1. $S$ ensures that the session is not expired by checking that the current time $t_{curr} < \text{EXP}$.
2. $S$ creates an initial $IV$. With the shared keys $\{s_i^f\}$, $S$ onion encrypts the data payload $M$ by setting $O_{l^f} = M$ and $IV_{l^f} = IV$ and computing the following for $i \leftarrow (l^f - 1)..0$:

$$\{O_i, IV_i\} = \text{ADD\_LAYER}(s_D, IV_{i+1}, O_{i+1}) \tag{19}$$

3. $S$ places $IV_0$ in the common header CHDR.
4. $S$ sends out the resulting data packet $\{\text{CHDR}, \text{AHDR}^f, O_0\}$.

**Processing by intermediate nodes.** Each intermediate node $n_i^f$ on the forward path processes a received data packet of the form $\{\text{CHDR}, \text{AHDR}^f, O\}$ with its local secret key $SV_i^f$ as follows:

1. $n_i^f$ retrieves the key $s_i^f$ shared with $S$ and the routing information $R_i^f$ from $\text{AHDR}^f$:

$$\{s_i^f, R_i^f, \text{AHDR}^{f\prime}\} = \text{PROC\_AHDR}(SV_i^f, \text{AHDR}^f) \tag{20}$$

   PROC_AHDR also verifies the integrity of AHDR, and checks that the session has not expired.
2. $n_i^f$ obtains $IV$ from CHDR and removes one layer of encryption from the data payload:

$$\{O', IV'\} = \text{REMOVE\_LAYER}(s_i^f, IV, O) \tag{21}$$

3. $n_i^f$ updates the IV field in CHDR with $IV'$.
4. $n_i^f$ sends the resulting packet $\{\text{CHDR}', \text{AHDR}^{f\prime}, O'\}$ to the next node according to $R_i^f$.

The above procedures show that the intermediate node processing requires only symmetric-cryptography operations.

**Destination processing.** $D$ processes incoming data packets as the intermediate nodes. Removing the last encryption layer from the onion payload $D$ obtains the original data payload $M$ sent by $S$. Additionally, for the first data packet $D$ retrieves $\text{AHDR}^b$ from the payload, and stores the $\{s_D, R_0^b, \text{AHDR}^b\}$ locally so that $D$ can retrieve $\text{AHDR}^b$ when it wishes to send packets back to $S$.

**Processing for the backward path.** Sending and processing a HORNET packet along the backward path is the same as that for the forward path, with the exception of processing involving the data payload. Because $D$ does not possess the symmetric keys that each node on the backward path shares with $S$, $D$ cannot onion-encrypt its payload. Therefore, instead of REMOVE_LAYER, $D$ and the intermediate nodes use ADD_LAYER to process the data payload, and the source node recovers the data with REMOVE_LAYER.

# 5. SECURITY ANALYSIS

This section describes how HORNET defends against well-known de-anonymization attacks and meets the design goals of Section 2.3. We also present defenses against denial of service attacks. A taxonomy of attacks against low-latency anonymity systems, as well as formal proofs showing that HORNET satisfies the correctness, security, and integrity properties defined by Camenisch and Lysyanskaya [17] are detailed in the full version [19].

## 5.1 Passive De-anonymization

**Session linkage.** Each session is established independently from every other session, based on fresh, randomly generated keys. Sessions are in particular not related to any long term secret or identifier of the host that creates them. Thus, two sessions from the same host are unlinkable, i.e., they are cryptographically indistinguishable from sessions of two different hosts.

**Forward/backward flow correlation.** The forward and backward headers are derived from distinct cryptographic keys and therefore cannot be linked. Only the destination is able to correlate forward and backward traffic, and could exploit this to discover the round-trip time (RTT) between the source and itself, which is common to all low-latency anonymity systems. Sources willing to thwart such RTT-based attacks from malicious destinations could introduce a response delay for additional protection.

**Packet correlation.** HORNET obfuscates packets at each hop. This prevents an adversary who observes packet bit patterns at two points on a path from linking packets between those two points. In addition to onion encryption, we also enforce this obfuscation

by padding the header and the payload to a fixed length, thwarting packet-size-based correlation.[5] While this does not prevent the adversary from discovering that the same flow is passing his observation points using traffic analysis, it makes this process non-trivial, and allows upper-layer protocols to take additional measures to hide traffic patterns. The hop-by-hop encryption of the payload also hides the contents of the communication in transit, protecting against information leaked by upper layer protocols that can be used to correlate packets.

**Path length and node position leakage.** HORNET protects against the leakage of a path's length and of the nodes' positions on the path (i.e., the relative distance, in hops, to the source and the destination). In the setup phase, this protection is guaranteed by Sphinx, so only the common header and FS Payload are subject to leakage (see Section 3.3 for the exact structure of the packets). It is straightforward to see that the common header does not contain path or position information. The FS Payload length is padded to the maximum size, and remains constant at each hop (see Algorithm 1). After adding its FS to the front of the FS Payload, each node re-encrypts the FS payload, making it infeasible for the next nodes to see how many FSes have previously been inserted.

During data transmission, neither the common header nor the data payload contain information about path length or node position, so only the AHDR (anonymous header) needs to be analyzed. The AHDR is padded to a maximum length with random bytes, and its length remains constant as it traverses the network (see Algorithm 3). The FSes contained in the AHDR are onion encrypted, as is the padding added at each hop. Thus, it is not possible to distinguish the initial random padding from the encrypted FSes, and neither of these from encrypted padding added by the nodes.

**Timing for position identification.** A malicious node could try to learn its position on the path of a session by measuring timing delays between itself and the source (or the destination) of that session. HORNET offers two possible countermeasures. In the first, we assume that the malicious node wishes to measure the network delay between itself and the source. To perform such a measurement, the node must observe a packet directed to the source (i.e., on the backward path) and then observe a response packet from the source (on the forward path). However, HORNET can use asymmetric paths [31], making this attack impossible if the single node is not on both forward and backward paths.

The second countermeasure is that, even if the node is on both paths, it is still non-trivial to discover that a specific forward flow corresponds to a certain backward flow, since the forwarding segments for the two paths are independent. To link the forward and backward flows together the node would need to rely on the traffic patterns induced by the upper-layer protocols that are running on top of HORNET in that session.

## 5.2 Active De-anonymization

**Session state modification.** The state of each node is included in an encrypted FS. During the session setup, the FSes are inserted into the FS payload, which allows the source to check the integrity of these FSes during the setup phase. During data transmission, FSes are integrity-protected as well through per-hop MACs computed by the source. In this case, each MAC protecting an FS is computed using a key contained in that FS. This construction is secure because every FS is encrypted using a PRP keyed with a secret value known only to the node that created the FS: if the FS is modified, the authentication key that the node obtains after decryption is a new pseudo-random key that the adversary cannot control. Thus, the probability of the adversary being able to forge a valid MAC is still negligible.

**Path modification.** The two HORNET data structures that hold paths (i.e., FS payloads in the setup phase and AHDRs), use chained per-hop MACs to protect path integrity and thwart attacks like inserting new nodes, changing the order of nodes, or splicing two paths. The source can check such chained per-hop MACs to detect the modifications in the FS payload before using the modified FS payload to construct AHDRs, and similarly intermediate nodes can detect modifications to AHDRs and drop the altered packets. These protections guarantee path information integrity as stated in Section 2.3.

**Replay attacks.** Replaying packets can facilitate some types of confirmation attacks [42]. For example, an adversary can replay packets with a pre-selected pattern and have a colluding node identify those packets downstream. HORNET offers replay protection through session expiration; replayed packets whose sessions have expired are immediately dropped. Replay of packets whose sessions are not yet expired is possible, but such malicious behavior can be detected by the end hosts. Storing counters at the end hosts and including them in the payload ensures that replays are recognizable. The risk of detection helps deter an adversary from using replays to conduct mass surveillance. Furthermore, volunteers can monitor the network, to detect malicious activity and potentially identify which nodes or group of nodes are likely to be misbehaving. Honest ASes could control their own nodes as part of an intrusion detection system.

## 5.3 Payload Protection

**Payload secrecy.** Data packet payloads are wrapped into one layer of encryption using the key shared between the source and the destination, both for packets sent by the source on the forward and for packets sent by the destination on the backward path (see Section 4.4.4). Assuming that the cryptographic primitives used are secure, the confidentiality of the payload is guaranteed as long as the destination is honest. In Section 7.3 we discuss the guarantees for perfect forward secrecy for the data payload.

**Payload tagging or tampering.** HORNET does not use per-hop MACs on the payload of data packets for efficiency and because the destination would not be able to create such MACs for the packets it sends (since the session keys of the nodes are known only to the source). The lack of integrity protection allows an adversary to tag payloads. Admittedly, the use of tagging, especially in conjunction with replay attacks, allows the adversary to improve the effectiveness of confirmation attacks. However, end-to-end MACs protect the integrity of the data, making such attacks (at a large scale) detectable by the end hosts.

## 5.4 Denial-of-Service (DoS) Resilience

**Computational DoS.** The use of asymmetric cryptography in the setup phase makes HORNET vulnerable to computational DoS attacks, where adversaries can attempt to deplete a victim node's computation capability by initiating a large number of sessions through this node. To mitigate this attack, HORNET nodes can require each client that initiates a session to solve a cryptographic puzzle [23] to defend against attackers with limited computation power. Alternatively, ISPs offering HORNET as a service can selectively allow connections from customers paying for the anonymity service.

**State-based DoS.** HORNET is not vulnerable to attacks where adversaries maintain a large number of active sessions through a victim node. One of HORNET's key features is that all state is carried

[5] A bandwidth-optimized alternative would be to allow two or three different payload sizes, at the cost of decreased anonymity.

within packets, thus no per-session memory is required on nodes or rendezvous points.

## 5.5 Topology-based Analysis

Unlike onion routing protocols that use global re-routing through overlay networks (e.g., Tor [25] and I2P [49]), HORNET uses short paths created by the underlying network architecture to reduce latency, and is therefore bound by the network's physical interconnection and ISP relationships. This is an unavoidable constraint for onion routing protocols built into the network layer [32, 44]. Thus, knowledge of the network topology enables an adversary to reduce the number of possible sources (and destinations) of a flow by only looking at the previous (and next) hop of that flow. For example, in Figure 3(a), assume that AS0 is controlled by a passive adversary. The topology indicates that any packet received from AS1 must have originated from a source located at one of {AS1, AS2, AS3, AS4, AS5}.

We evaluate the information leakage due to the above topology constraints in the scenario where a single AS is compromised. We derive AS-level paths from iPlane trace-route data [6], and use AS-level topology data from CAIDA [36]. For each AS on each path we assume that the AS is compromised and receives packets from a victim end host through that path. We compute the end host's anonymity set size learned by the adversary according to the topology. For instance, in Figure 3(a), if AS0 is compromised and receives from AS1 packets originally sent by a user in AS4, we compute the size of the anonymity set composed of all the ASes that can establish valley-free paths traversing the link from AS1 to AS0. In this example, the anonymity set size would be the sum of the sizes of AS1, AS2, AS3, AS4, and AS5.

Similar to Hsiao et al. [32], we use the number of IPv4 addresses to estimate the size of each AS. Figure 3(b) plots the CDF of the anonymity set size for different distances (in number of AS hops) between the adversary and the victim end host. For adversarial ASes that are 4 hops away, the anonymity set size is larger than $2^{31}$ in 80% of the cases. Note that the maximum anonymity set size is $2^{32}$ in our analysis, because we consider only IPv4 addresses.

**Implications of path knowledge.** Knowledge about the path, including the total length of the path and an adversarial node's position on the path, significantly downgrades the anonymity of end hosts. Considering again Figure 3(a), if the adversary controlling AS0 sees a packet incoming from AS1 and knows that it is 4 hops away from the source host, he learns that the source host is in AS4. Compared with the previous case, we see that the anonymity set size is strongly reduced.

We quantify additional information leakage in the same setting as the previous evaluation. Figure 3(c) represents the CDFs of the anonymity set sizes of end hosts according to the distance to the compromised AS. The anonymity set sizes are below $2^{28}$ in 90% of the cases when the adversarial ASes are 4 hops away, with an average size of $2^{23}$. This average size decreases to $2^{17}$ for the cases where the adversarial ASes are 7 hops away from the target hosts.

Previous path-based anonymity systems designed for the network layer either fail to hide knowledge about the path [44] or only partially obscure the information [32]. In comparison, HORNET protects both the path length and the position of each node on the path, which significantly increases the anonymity-set size.

# 6. EVALUATION

We implemented the HORNET router logic in an Intel software router using the Data Plane Development Kit (DPDK) [4]. To our knowledge, no other anonymity protocols have been implemented in a router SDK. We also implemented the HORNET client in Python. Furthermore, we assembled a custom crypto library based on the Intel AESNI cryptographic library [5], the curve25519-donna library [3], and the PolarSSL libraries [8]. We use IP forwarding in DPDK as our performance baseline. For comparison, we implemented the data forwarding logic from Sphinx, LAP, Dovetail, and Tor using DPDK and our cryptographic library.

| Scheme | Header Length | Sample Length (Bytes) |
|---|---|---|
| LAP | $12 + 2s \cdot r$ | 236 |
| Dovetail | $12 + s \cdot r$ | 124 |
| Sphinx | $32 + (2r + 2)s$ | 296 |
| Tor | $3 + 11 \cdot r$ | 80 |
| HORNET | $8 + 3r \cdot s$ | 344 |

**Table 2: Comparison between the length of different packet header formats in bytes. $s$ is the length of symmetric elements and $r$ is the maximum AS path length. For the sample length, we select $s = 16$ Bytes and $r = 7$. Analysis of iPlane paths shows that more than 99% of all paths have fewer than 7 AS hops.**

Fairly comparing the performance of anonymity systems at the application layer with those that operate at the network layer is challenging. To avoid penalizing Tor with additional propagation delay caused by longer paths and processing delay from the kernel's network stack, we implemented Tor at the network layer (as suggested by Liu et al. [35]). Tor's design requires relay nodes to perform SSL/TLS and transport control. SSL/TLS between neighboring relays at the application layer maps to link encryption between neighboring nodes at the network layer, which we consider orthogonal but complementary to HORNET (see Section 7.2). Hence, for fair comparison, we implemented the network-layer Tor without SSL/TLS or transport control logic. Throughout our evaluation we refer to this implementation of Tor as L3 Tor.

Our testbed contains an Intel software router connected to a Spirent TestCenter packet generator and analyzer [10]. The software router runs DPDK 1.7.1 and is equipped with an Intel Xeon E5-2680 processor (2.70 GHz, 2 sockets, 16 logical cores/socket), 64 GB DRAM, and 3 Intel 82599ES 40 Gb/s network cards (each with 4 10 Gb/s ports). We configured DPDK to use 2 receiving queues for each port with 1 adjacent logical core per queue.

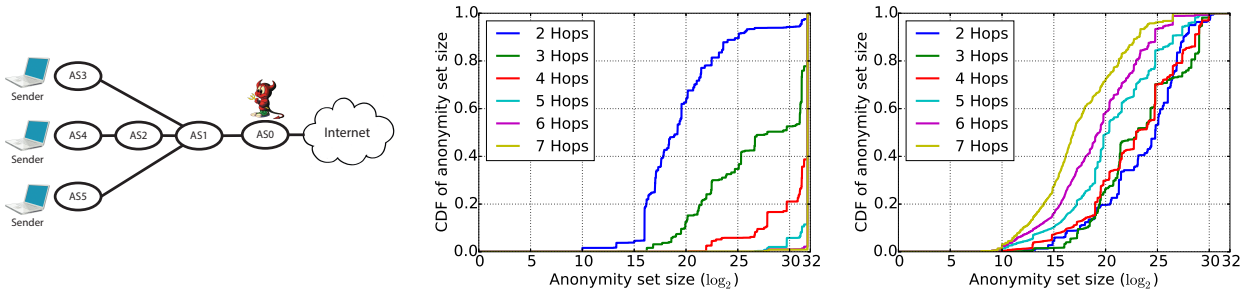
## 6.1 Data Forwarding Performance

**Forwarding latency.** We measure the CPU cycles consumed to forward a data packet in all schemes. Figure 4 shows the average latency (with error bars) to process and forward a single data packet in all schemes (except Sphinx[6]) when payload sizes vary. We observe that HORNET, even with onion encryption/decryption over the entire payload and extensive header manipulation, is only 5% slower than LAP and Dovetail for small payloads (64 bytes). For large payloads (1200 bytes[7]), HORNET is 71% slower (about 400 nanoseconds slower per packet when using a single core) than LAP and Dovetail. However, the additional processing overhead enables stronger security guarantees.

**Header overhead.** As a result of carrying anonymous session state (specifically cryptographic keys) within packet headers, HORNET

[6]We omit Sphinx from the comparison for better readability. In our experiments, processing a Sphinx packet takes more than 640K cycles due to asymmetric cryptographic operations. This is 3 orders of magnitude slower than that of HORNET, L3 Tor, LAP, and Dovetail.

[7]Because LAP, Dovetail, and HORNET all have large packet headers of 300+ bytes, we limit the largest payload in our experiments to be 1200 bytes.




(a) (b) Without path knowledge (c) With path knowledge

**Figure 3: a) An example AS-level topology with an adversarial AS (AS0). b) CDF of anonymity-set size when a position-agnostic AS on path is adversarial. "Hops" indicates the number of ASes between the adversarial AS and the victim end host. For example, the point (25, 0.4) on the line "3 hops" means that the anonymity set size is smaller than $2^{25}$ in 40% of cases when the end host is 3 hops away from the adversarial AS. c) CDF of anonymity-set size when an adversarial AS knows its own position on the path. For Figures b) and c), the maximum size of an end host's anonymity set is $2^{32}$ because we consider the IPv4 address space. Therefore, the ideal case for an end host is that the anonymity set size is $2^{32}$ with probability equal to 1.**

headers are larger than Sphinx, L3 Tor, LAP, and Dovetail headers (see Table 2). While larger headers reduce net throughput (i.e., goodput), this tradeoff appears acceptable: compared to L3 Tor, no state is required at relay nodes, enabling scalability; compared to Sphinx, data processing speed is higher; compared to LAP and Dovetail, HORNET provides stronger security properties.

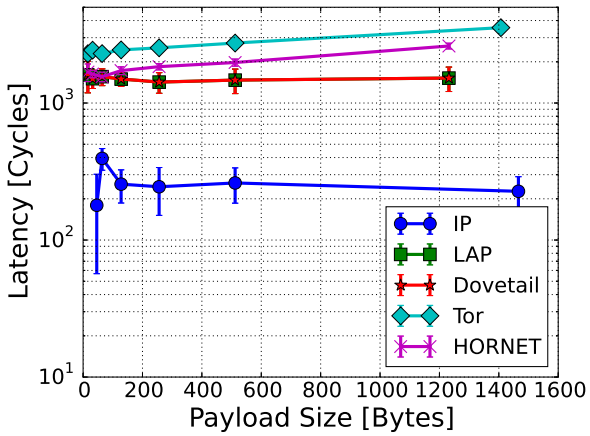



**Figure 4: Per-node data forwarding latency on a 10 Gbps link. Lower is better.**

**Goodput.** We further compare all the schemes by goodput, which excludes the header overhead from total throughput. Goodput is a comprehensive metric to evaluate both the packet processing speed and protocol overhead. For example, a scheme where headers take up a large proportion of packets yields only low goodput. On the other hand, a scheme with low processing speed also results in poor goodput.

Figure 5(a) and Figure 5(b) demonstrate the goodput of all schemes (except Sphinx[8]) on a 10 Gb/s link when varying the number of hops $r$, with 40-byte and 1024-byte payloads, respectively. Larger $r$ means larger header sizes, which reduces the resulting goodput.

When the payload size is small, the goodput of all protocols remains stable. This is due to the fact that no scheme can saturate the link, and accordingly the goodput differences between the three schemes mainly reflect the different processing latencies among them. Consequently, L3 Tor's and HORNET's goodput is 32% less than that of LAP and Dovetail. On the other hand, when the payload size is large, all schemes except Sphinx can saturate the 10 Gb/s link. HORNET can reach 87% of LAP's goodput while providing stronger security guarantees.

## 6.2 Max Throughput on a Single Router

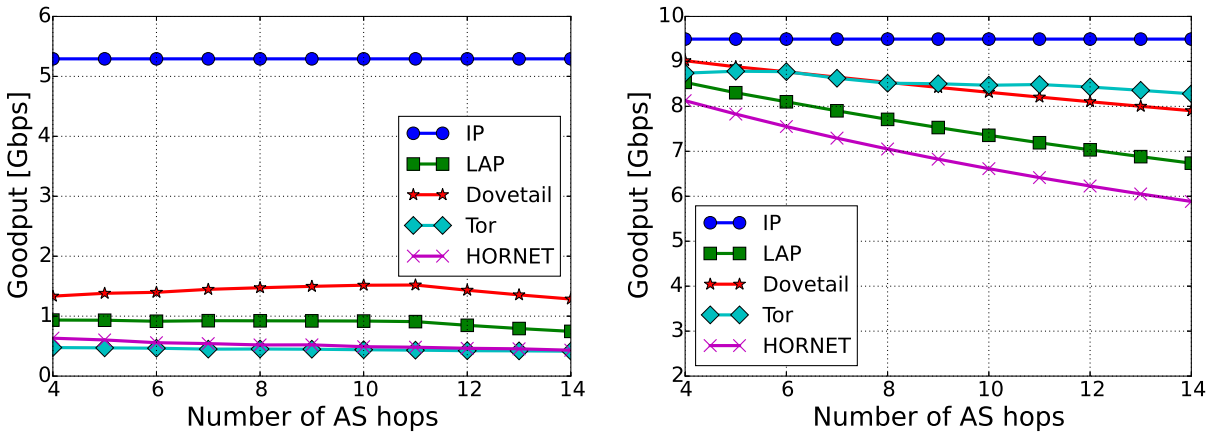
To investigate how our implementation scales with respect to the number of CPU cores, we use all 12 ports on the software router, generating HORNET data packets at 10 Gb/s on each port. Each packet contains a 7 AS-hop header and a payload of 512 bytes, and is distributed uniformly among the working ports. We monitor the aggregate throughput on the software router.

The maximal aggregate throughput of HORNET forwarding in our software router is 93.5 Gb/s, which is comparable to today's switching capacity of a commercial edge router [1]. When the number of cores ranges from 1 to 4, our HORNET implementation can achieve full line rate (i.e., 10 Gb/s per port). As the number of cores increases to 5 and above, each additional port adds an extra 6.8Gb/s.

## 6.3 Session Setup Performance

We evaluate the latency introduced by processing setup packets on each border router. Similar to measuring the latency of data forwarding, we also instrument the code to measure CPU cycles consumed to process packets in the session setup phase. Table 3 lists the average per-node latency for processing the two setup packets in HORNET's session setup phase. Due to a Diffie-Hellman key exchange, processing the two setup packets in the session setup phase increases processing latency (by about 240$\mu$s) compared to data packet processing. However, HORNET must only incur this latency once per session.

[8]Sphinx's goodput is less than 10 Mb/s in both cases because of its large packet headers and asymmetric cryptography for packet processing.




(a) 40 Byte payloads

(b) 1024 Byte payloads

**Figure 5: a) Data forwarding goodput on a 10 Gbps link for small packets (40 Byte payloads); b) Data forwarding goodput large packets (1024 Byte payloads). Higher is better.**

| Packet | Latency (K cycles) | Latency ($\mu$s) |
|---|---|---|
| P❶ | 661.95 $\pm$ 30.35 | 245.17 $\pm$ 11.24 |
| P❷ | 655.85 $\pm$ 34.03 | 242.91 $\pm$ 12.60 |

**Table 3: Per-node latency to process session setup packets with standard errors.**

# 7. DISCUSSION

## 7.1 Retrieving Paths Anonymously in FIAs

HORNET assumes that the source can obtain a forward path and a backward path to an intended destination anonymously in FIAs. We briefly discuss how a source host using HORNET can retrieve two such paths in NIRA, SCION and Pathlets.

SCION hosts rely on path servers to retrieve paths. In SCION, each destination node registers on a central server its "half" path: the path to/from the network "core". To compose full paths (forward and backward paths) between a source and a destination, the source only needs to anonymously fetch the destination's half paths from/to the network core and combine them with its own half paths.

To anonymously retrieve a destination's half paths, the source can use one of the following two methods. As a first method, the source can obtain the path to/from a path server through an unprotected query using other schemes, from resolver configuration, or from local services similar to DHCP. The source then establishes an anonymous HORNET session to the server. Once a HORNET session is created, the source can proceed to anonymously request half paths of the destination. Though it is possible to reuse the established HORNET session to a path server to query multiple paths (for different destinations) for better efficiency, using a separate session to retrieve each path is more secure because it prevents profiling attacks.

Alternatively, the source can leverage a private information retrieval (PIR) scheme [20] to retrieve the path anonymously from the path server, so that the path server cannot distinguish which destination the source connects to. However, a PIR scheme will inevitably add bandwidth and computational overhead to both the source and the path server, increasing session setup phase latency [37].

In NIRA and Pathlets, the situation is different because routing information (i.e., inter-domain addresses and route segments, and pathlets, respectively) is disseminated to users. The source can therefore keep a database local path database, querying it (locally) on demand.

## 7.2 Integrating with Security Mechanisms at Different Layers

At the network layer, HORNET can benefit from ASes that offer traffic redirection to mitigate topology-based attacks (see Section 5.5). For instance, ASes can allow paths that deviate from the valley-freeness policy to increase the anonymity set size of end hosts. This enables a trade-off between path length and anonymity, as described by Sankey and Wright [44].

In addition, upper-layer anonymity protocols can be used in conjunction with HORNET to provide stronger anonymity guarantees. For example, to entirely remove the concerns of topology-based attacks, a single-hop proxy or virtual private network (VPN) could be used to increase the size of the anonymity sets of end hosts. Similar solutions could also protect against upper-layer de-anonymization attacks, in particular fingerprinting attacks on the transport protocol [46].

At lower layers, HORNET is also compatible with link-layer protection such as link-level encryption. The role of link-level encryption in HORNET is comparable to SSL/TLS in Tor. Link encryption prevents an adversary eavesdropping on a link from being able to distinguish individual sessions from each other, therefore making confirmation attacks much harder for this type of adversary.

## 7.3 Limitations

**Targeted confirmation attacks.** When for a certain session an adversary controls both the node closest to the source and the node closest to the destination (or the destination itself), it can launch confirmation attacks by analyzing flow dynamics.These attacks can be made more effective by replaying packets.

HORNET, like other low-latency onion routing schemes [25], cannot prevent such confirmation attacks targeting a small number of specific users [45, 33]. However, HORNET raises the bar of deploying such attacks at scale: the adversary must be capable of

controlling a significant percentage of ISPs often residing in multiple geopolitical areas. In addition, the packet obfuscation measures built into HORNET (discussed in Section 5) make it non-trivial to link two flows, since it is not possible to simply match packets through bit patterns. Timing intervals for packet sequences need to be stored and compared, thus performing such operations for a large fraction of the observed flows is expensive. Furthermore, it is difficult for attackers to perform active attacks (e.g., packet replay) at scale while remaining undetected. For instance, a downstream benign AS can detect replayed packets by a compromised upstream AS; end hosts can also detect and report packet tagging attacks when (a threshold number of) end-to-end MACs do not successfully verify.

**Perfect forward secrecy.** A drawback of HORNET's efficiency-driven design is that it does not provide perfect forward secrecy for the link between communicating parties. This means that an adversary could record the observed traffic (the setup phases, in particular), and if it later compromises a node, it learns which node was next on the path for each recorded session. This is an unavoidable limitation of having a setup that consists of a single round-trip.

Other systems (e.g., Tor) use a telescopic setup[9], which achieves perfect forward secrecy at the cost of diminished performance (in particular higher latency, and also an additional asymmetric cryptographic operation per node). Using a telescopic setup is also possible for HORNET, but in addition to the performance cost it also requires that all paths be reversible. However, this requirement does not hold in today's Internet, where a significant fraction of AS-level paths are asymmetric [31].

It is important to note that in HORNET it is still possible to achieve perfect forward secrecy for the contents of the communication, i.e., for the data exchanged between sources and destinations. The destination needs to generate an ephemeral Diffie-Hellman key pair, and derive an additional shared key from it.[10] Destinations also need to generate a new local secret $SV$ frequently, so in the event of a destination being compromised it is not possible for the adversary to decrypt FSes used in expired sessions.

# 8. RELATED WORK

**Anonymity systems as overlays.** The study of anonymous communication began with Chaum's proposal for mix networks [18]. A number of message-based mix systems have been proposed and deployed since [30, 38, 21, 22]. These systems can withstand an active adversary and a large fraction of compromised relays, but rely on expensive asymmetric primitives, and message batching and mixing. Thus, they suffer from large computational overhead and high latency.

Onion routing systems [43, 14, 15, 25] were proposed to efficiently support interactive traffic. In general, low-latency onion routing systems are vulnerable to end-to-end confirmation attacks [34], and may fail to provide relationship anonymity when two routers on the path are compromised [29, 33]. HORNET shares these limitations.

One specific onion routing system, Tor, has a number of security advantages over HORNET. Tor can prevent replays and has perfect forward secrecy for its sessions. Additionally, due to its overlay design which uses global redirection, Tor is not constrained by the underlying network topology. However, global redirection enables the attack vector that allows even single compromised ASes to perform confirmation attacks [40, 13], as one AS can be traversed multiple times. This attack is not possible in HORNET since packets traverse each AS on the path only once.

In addition, HORNET's performance also distinguishes it from all existing schemes based on overlay networks: first, HORNET can directly use short paths provided by underlying network architectures, reducing propagation latency; second, HORNET requires only a single round trip to establish a session, reducing the setup delay; third, HORNET eliminates the processing and queuing delays both on relay nodes and in the kernel's network stack; finally, edge routers in HORNET offer higher throughput compared to voluntarily-contributed end hosts, increasing the total throughput of anonymous traffic.

**Anonymity systems in FIAs.** Hsiao et al. [32] explored the design space of efficient anonymous systems with a relaxed adversary model. In their scheme, LAP, the adversary can compromise only a single node, and the first hop must always be honest. Sankey and Wright proposed Dovetail [44] (based on Pathlets [28] and SCION [51, 12]) which has the same attacker model as LAP, except it allows the first hop to be compromised. Moreover, neither LAP nor Dovetail can support asymmetric paths where packets traverse different sets of nodes in different directions. HORNET offers three improvements over LAP and Dovetail: 1) HORNET fully hides path information, i.e., total path length and nodes' positions, in packet headers; 2) HORNET protects and obfuscates packet contents by onion-encryption/decryption, thwarting correlating packets of the same flow by selectors; 3) HORNET supports asymmetric paths and allows the first hop ASes to be compromised. Though HORNET introduces additional overhead in comparison with LAP and Dovetail, our evaluation results show that HORNET can still support high-speed packet forwarding at nearly 80% of line rate.

The research community has also explored applying onion routing to FIAs. Liu et al. [35] proposed Tor instead of IP as an FIA that regards anonymity as the principal requirement for the network architecture. However, details on how to scale Tor's current design (requiring per-circuit state) to Internet scale were not addressed.

DiBenedetto et al. [24] proposed ANDaNA, to enable onion routing in Named Data Networking (NDN) [50]. NDN focuses on content delivery and thus inherently different from the FIAs we considered.

# 9. CONCLUSION

In this paper, we address the question of "what minimal mechanism can we use to frustrate pervasive surveillance?" and study the design of a high-speed anonymity system supported by the network architecture. We propose HORNET, a scalable and high-speed onion routing scheme for future Internet architectures. HORNET nodes can process anonymous traffic at over 93 Gb/s and require no per-flow state, paving the path for Internet-scale anonymity. Our experiments show that small trade-offs in packet header size greatly benefit security, while retaining high performance.

# 10. ACKNOWLEDGMENTS

We would like to thank our shepherd Prateek Mittal, and the anonymous CCS reviewers for their suggestions for improving the paper. We also grateful for insightful discussions with Ian Goldberg and the members of the ETH Zürich Network Security group for their discussions and feedback.

The research leading to these results received funding from the European Research Council under the European Union's Seventh

[9] In the telescopic setup, a source iteratively sets up a shared key with each AS: the source sets up a shared key with the first-hop AS; the source sets up a shared key with the $n$th-hop AS through the channel through 1st-hop AS to $(n-1)$th-hop AS.

[10] This feature, though omitted in Section 4 for simplicity, is part of our implementation. It is done in such a way that the forward secret shared key is included in the destination's FS during the setup, without any additional packet being required.

Framework Programme (FP7/2007-2013) / ERC grant agreement 617605. George Danezis is supported by the EU H2020 Project PANORAMIX (653497) and EPSRC Project on "Strengthening anonymity in messaging systems" (EP/M013286/1). We also gratefully acknowledge support by ETH Zürich, and by Intel for their equipment donation that enabled the high-performance experiments.

# 11. REFERENCES

[1] Cisco ASR-1000. `http://www.cisco.com/c/en/us/products/routers/`. Retrieved on 2015.04.28.

[2] Cisco routers. `http://www.cisco.com/c/en/us/products/routers`. Retrieved on 2015.08.05.

[3] curve25519-donna. `https://code.google.com/p/curve25519-donna/`. Retrieved on 2014.12.13.

[4] DPDK: Data plane development kit. `http://dpdk.org/`. Retrieved on 2014.12.23.

[5] Intel AESNI sample library. `https://software.intel.com/en-us/articles/download-the-intel-aesni-sample-library`. Retrieved on 2014.12.13.

[6] iPlane dataset. `http://iplane.cs.washington.edu/data/data.html`. Traceroute data was generated on October 12, 2014.

[7] NSA targets the privacy-conscious. `http://daserste.ndr.de/panorama/aktuell/NSA-targets-the-privacy-conscious,nsa230.html`. Retrieved on 2015.05.13.

[8] PolarSSL. `https://polarssl.org/`. Retrieved on 2014.12.13.

[9] Segment routing architecture (IETF draft). `https://datatracker.ietf.org/doc/draft-ietf-spring-segment-routing/`. Retrieved on 2015.05.13.

[10] Spirent TestCenter. `http://www.spirent.com/Ethernet_Testing/Software/TestCenter`. Retrieved on 2014.12.23.

[11] Tor metrics. `https://metrics.torproject.org`. Retrieved on 2015.05.13.

[12] David Barrera, Raphael M. Reischuk, Pawel Szalachowski, and Adrian Perrig. SCION Five Years Later: Revisiting Scalability, Control, and Isolation on Next-Generation Networks. *arXiv/1508.01651*, August 2015.

[13] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *ACM WPES*, 2007.

[14] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture, 2000. White paper, Zero Knowledge Systems, Inc.

[15] Zach Brown. Cebolla: Pragmatic IP anonymity. In *Ottawa Linux Symposium*, 2002.

[16] R. Bush and R. Austein. The resource public key infrastructure (RPKI) to router protocol. IETF RFC 6810.

[17] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In *CRYPTO*, 2005.

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

[19] Chen Chen, Daniele Enrico Asoni, David Barrera, George Danezis, and Adrian Perrig. HORNET: High-speed Onion Routing at the Network Layer. *arXiv/1507.05724*, July 2015.

[20] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6), 1998.

[21] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *IEEE S&P*, 2003.

[22] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *IEEE S&P*, 2009.

[23] Drew Dean and Adam Stubblefield. Using client puzzles to protect TLS. In *USENIX Security*, 2001.

[24] Steven DiBenedetto, Paolo Gasti, Gene Tsudik, and Ersin Uzun. ANDaNA : Anonymous named data networking application. In *NDSS*, 2011.

[25] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

[26] S. Farrell and H. Tschofenig. Pervasive monitoring is an attack. IETF RFC 7258.

[27] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.

[28] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. *ACM SIGCOMM*, 2009.

[29] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *ACM Information Hiding (IH) Conference*, 1996.

[30] Ceki Gülcü and Gene Tsudik. Mixing email with Babel. In *NDSS*, 1996.

[31] Yihua He, Michalis Faloutsos, Srikanth Krishnamurthy, and Bradley Huffaker. On routing asymmetry in the Internet. In *IEEE GLOBECOM*, 2005.

[32] Hsu Chun Hsiao, Tiffany Hyun Jin Kim, Adrian Perrig, Akira Yamada, Samuel C. Nelson, Marco Gruteser, and Wei Meng. LAP: Lightweight anonymity and privacy. In *IEEE S&P*, 2012.

[33] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul F. Syverson. Users get routed: traffic correlation on Tor by realistic adversaries. In *ACM CCS*, 2013.

[34] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing attacks in low-latency mix-based systems. In *FC*, 2004.

[35] Vincent Liu, Seungyeop Han, Arvind Krishnamurthy, and Thomas Anderson. Tor instead of IP. In *ACM HotNets*, 2011.

[36] P. Mahadevan, D. Krioukov, M. Fomenkov, B. Huffaker, X. Dimitropoulos, K. Claffy, and A. Vahdat. The Internet AS-level topology: Three data sources and one definitive metric. In *ACM SIGCOMM*, 2006.

[37] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In *USENIX Security*, 2011.

[38] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster protocol v. 2. IETF Draft, 2003.

[39] R. Moskowitz and P. Nikander. Host identity protocol (HIP) architecture. IETF RFC 4423.

[40] Steven J. Murdoch and Piotr Zieliński. Sampled traffic analysis by Internet-Exchange-level adversaries. In *PETS*, 2007.

[41] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity - a proposal for terminology. In *Designing Privacy Enhancing Technologies*, 2001.

[42] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies*, 2001.

[43] Michael G. Reed, Paul F. Syverson, and M. Goldschlag David. Anonymous connections and onion routing. *IEEE JSAC*, 1998.

[44] Jody Sankey and Matthew Wright. Dovetail: Stronger anonymity in next-generation internet routing. In *PETS*, 2014.

[45] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In *ESORICS*, 2003.

[46] Matthew Smart, G. Robert Malan, and Farnam Jahanian. Defeating TCP/IP stack fingerprinting. In *USENIX Security*, 2000.

[47] Wei Wang, Mehul Motani, and Vikram Srinivasan. Dependent link padding algorithms for low latency anonymity systems. In *ACM CCS*, 2008.

[48] Xiaowei Yang, David Clark, and Arthur W Berger. NIRA: a new inter-domain routing architecture. *IEEE/ACM Transactions on Networking*, 2007.

[49] Bassam Zantout and Ramzi Haraty. I2P data communication system. In *ICN*, 2011.

[50] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Kimberley Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. In *ACM SIGCOMM*, 2014.

[51] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. SCION: Scalability, control, and isolation on next-generation networks. In *IEEE S&P*, 2011.