SPINE: Surveillance Protection in the Network Elements

Trisha Datta, Nick Feamster, Jennifer Rexford, and Liang Wang
Princeton University

Abstract

Internet Protocol (IP) addresses can reveal information about communicating Internet users and devices, even when the rest of the traffic between them is encrypted. At the same time, IP addresses serve as endpoints for network-layer communication and, as a result, are typically visible to the intermediate routers to allow them to forward traffic to its ultimate destination. Previous approaches to obfuscate the IP addresses of the sender and receiver commonly depend on either custom user software (e.g., Tor browser) or significant modifications to network hardware along the end-to-end path (which has proved to be a major roadblock). SPINE, on the other hand, conceals IP addresses and relevant TCP fields from intermediate—and potentially adversarial—autonomous systems (ASes) but requires only two participating ASes and no cooperation from end hosts. To demonstrate SPINE's practicality, we have implemented it on commodity programmable switches using the P4 programming language. Our evaluation shows that SPINE can run at hardware rates on commodity switches, paving the way to real-world deployment.

1 Introduction

Internet traffic relies on the Internet Protocol (IP) to transmit data between endpoints: each Internet data packet contains an IP source and IP destination address. Network devices use IP addresses to forward traffic to Internet endpoints, but at the same time, IP addresses can reveal information about users and devices and the destinations with which they are communicating. Thus, even if Internet traffic is encrypted, IP addresses can still allow eavesdroppers to determine who is communicating with whom [16]. This threat becomes more acute when traffic traverses untrusted or adversarial domains, including foreign governments, public networks, or untrusted transit ISPs or carrier networks.

The threat of surveillance by intermediate networks is clear and present for many countries and organizations. Brazil, for instance, built an undersea cable to Portugal to prevent traffic destined for Portugal from travelling through the United States [11]. Many countries rely on Western nations for Internet content [9], potentially raising these types of concerns for other countries. More generally, any network's ability to observe source and destination IP addresses puts privacy at risk by revealing the endpoints of the communications.

Previous work recognizes this privacy threat [4–6, 10, 19], but existing approaches face significant deployment challenges. IPSEC tunnels, for instance, provide protection by completely encrypting original IP packets. However, such operations are too expensive to deploy in the core of the Internet and redundant if payloads are already encrypted with TLS. Another well-studied approach is network-layer anonymity systems, which typically require many nodes along a traffic path to provide protection. A common technique, for example, is onion routing, which normally requires several nodes along an end-to-end path to participate in the protocol. These nodes often must both cooperate in the protocol *and* be located in autonomous systems that are not adversarial; past work on systems such as Tor [8] has shown that this constraint can often be impractical. Even in threat models where the adversary can control some fraction of the nodes, it is highly unlikely that all ASes involved in forwarding the traffic will agree to implement these protocols. Many of these systems *also* require end-users to participate in the protocol, further raising the barrier to deployment.

This paper surmounts these practical challenges, thwarting an adversary that controls an intermediate portion of the end-to-end path and wants to observe the true source and destination IP addresses of the traffic. Our approach, Surveillance Protection in the Network Elements (SPINE), encrypts IP addresses in packet headers before these packets enter the intermediate ASes and decrypts IP addresses when the packets exit the intermediate ASes. When packets travel through the intermediate ASes, these networks only see encrypted addresses in the headers, not the true source and destination IP addresses of the traffic. SPINE also encrypts TCP sequence and acknowledgment numbers (when available) to prevent the adversary from recognizing which packets belong to the same TCP flow. Unlike previous approaches, SPINE *only* requires

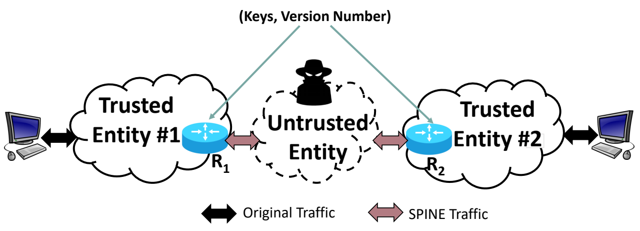

Figure 1: SPINE Setup

the cooperation of participating ASes (e.g., Trusted Entity #1 and #2 in Figure 1)—the behavior of end hosts and other ASes along the path can remain unchanged. Furthermore, because SPINE can be implemented on commodity programmable switches using the P4 language, it processes traffic at switch hardware rates, even in the core of the Internet.

SPINE exploits several recent technologies and trends. First, because packet payloads are often encrypted (e.g., using TLS), we need only worry about obfuscating certain IP and TCP header fields. Second, the rise of programmable switches based on the Protocol-Independent Switch Architecture (PISA) presents an opportunity to design protocols that manipulate arbitrary parts of packets (including the IP and TCP headers) while still processing packets at switch hardware rates. Third, we take advantage of the increasing ubiquity of IPv6 in the network core; IPv6 addresses have 128 bits, offering enough room to store metadata for specific types of encryption over IPv4 addresses. Currently, SPINE only protects privacy for IPv4 traffic, but future work will explore extensions to end-hosts communicating using IPv6.

Developing SPINE required us to tackle several challenging technical problems, including ensuring that routing still functions after encrypting the IP addresses, coping with the memory and computational constraints of modern programmable switches, and communicating encryption metadata (e.g., keys) across an untrusted network. The rest of the paper explains how we tackle these problems.

2 Practical Surveillance Protection

SPINE preserves the privacy of communicating hosts between two trusted network entities with minimal performance degradation. In the context of this high-level goal, we define the threat model and detail the design goals for SPINE.

2.1 Surveillance Threat Model

Figure 1 shows the problem setup. Two unmodified hosts communicate through two trusted entities—typically two ASes or two ISPs, yet these ASes need not be distinct or even independent. They could, in fact, be two topologically disjoint fragments of the *same* AS. The adversary resides in between these two entities and can observe all traffic that traverses the intermediate part (or parts) of the network that it controls. There may, in fact, be multiple (uncoordinated or colluding) adversaries between the two trusted ASes. The adversary may be an AS; it could also be *any* intermediate network point that an adversary controls, such as a public Internet exchange point (IXP) or even simply a link that the adversary controls or has access to. The goal of the adversary is to recover the actual source and destination IP addresses of network traffic it observes, given the contents of the packets that it can see.

In addition to passive observation, we assume that the adversary can also perform certain active attacks. For example, we assume that the adversary can send traffic to and from the trusted entities, as users of the respective trusted entities. We assume, however, that it cannot observe other users' traffic in the trusted entities' networks. Thus, certain chosen-plaintext attacks are feasible, whereby an attacker sends a packet with source address s and destination address d and can observe the corresponding encrypted IP addresses $E(s)$ and $E(d)$. On the other hand, we assume that the attacker cannot observe the same information for packets sent by another host.

Finally, we assume that the adversary is capable of performing timing analysis or fingerprinting attacks on traffic, which it might be able to use to identify destinations (or users) solely based on the characteristics of the encrypted traffic; for this reason, we also aim to prevent the adversary from attributing groups of packets to the same flow.

2.2 Practical Deployment Constraints

To make deployment practical, our design goals are as follows:

- **No cooperation from intermediate ASes:** Our solution should only require participating ASes (i.e., the two trusted entities) to make changes in order to be fully implemented. Only the parties who offer the service, or only the parties with the most incentives to participate, will need to participate.
- **No involvement from end users:** Our solution should not require end hosts to deploy SPINE. This makes the solution easier to use and gives users plausible deniability. While end users often lack knowledge about privacy risks and potential solutions, ASes and AS owners have both incentives to protect their customers' privacy and the technical ability to do so.
- **Processing at switch hardware rates:** We do not want SPINE to slow down traffic rates. To achieve this goal, we devise a solution that is implementable in the data plane of commodity programmable switches using the P4 language [2, 15]. The data plane is where decisions about how to forward packets take place, and P4 is a software-defined networking language that allows us to program such decisions so that they can be made at switch hardware rates. Because programmable switches are being deployed in edge networks, creating a solution that can be implemented on such switches facilitates the deployment of SPINE.

The commodity programmable switches we consider in this work are based on PISA [3, 17], as discussed in more detail in Section 4. PISA defines a processing pipeline for the data plane, providing a hardware-independent programming model for programmable switches. Although PISA switches allow us to write P4 programs for the data plane that run at switch hardware rates, this performance guarantee limits the amount and kind of packet processing PISA switches can perform (e.g., loops are not allowed, the switches have limited memory for the processing pipeline, etc.).

3 SPINE Design

Because the threat model is primarily concerned with adversaries who want to recover true source and destination IP addresses, SPINE aims to obfuscate these addresses. SPINE also prevents the adversary from attributing groups of packets to the same flow by additionally obfuscating TCP sequence and acknowledgment numbers when present. The main challenge of choosing an obfuscation technique is picking one that can be performed at switch hardware rates; Section 3.1 describes the encryption technique SPINE uses given this performance constraint. SPINE also enables periodic changes of the keys used for encryption so that even if an adversary steals a key, this knowledge will eventually become useless. Section 3.2 describes how we deal with packets in flight when keys change. Because routing depends on IP addresses, simply encrypting IP addresses would prevent packets from being forwarded successfully. Section 3.3 explains how we replace IPv4 headers with IPv6 headers to ensure that routing is still successful despite using encrypted IP addresses.

3.1 Efficient Cryptography on IP Addresses

SPINE uses the following approach to encrypt an IP address:

$$E(ip,k) = ip \oplus H(k, nonce)$$

where ip is the IPv4 address being encrypted, k is a secret key, $\oplus$ is the XOR operation, H is a keyed hash function, and *nonce* is a randomly-generated bit string. All SPINE routers in the two trusted entities (e.g., R_1 and R_2 in Figure 1) use the same k for encryption so that packets can exit the sending entity at one of several exit points and enter the receiving entity at one of several entry points (in contrast to IPSEC tunnels which establish point-to-point channels). We rotate the key k, discussed further in Section 3.2. The nonce is public and is sent along with the encrypted IP address in packet headers. The decrypting party can easily generate the same one-time pad to decrypt the IP address using the same secret key and the hash function. Because we assume that the attacker can send a packet with source address s and destination address d and observe the encrypted addresses $E(s)$ and $E(d)$, we choose H to be a cryptographically strong pseudorandom function (PRF). This means that after seeing $H(x)$ for all $x \in X$, an attacker will be unable to guess $H(y)$ for $y \notin X$ [1]. If present, TCP sequence and acknowledgment numbers are encrypted with the same H and *nonce* used to encrypt the IP addresses but with a different shared key k.

SPINE uses a one-time-pad-based encryption scheme instead of more standard encryption techniques because we want SPINE to run at switch hardware rates, and the memory and computational power required in standard encryption techniques are incompatible with this performance goal. We are thus limited to fairly simple cryptography operations, like the efficient single XOR operation required by one-time pad encryption. One big challenge of one-time pad encryption is facilitating the exchange of pads. SPINE overcomes this challenge by having a central coordinator that communicates with the trusted network entities and facilitates the distribution of secret keys to generate the pads.

One-time pad encryption is secure as long as the pads are random and not repeated [13]. Because we use a PRF, randomly generate nonces, and frequently switch keys for the hash function, this requirement should hold in practice. The main attack an adversary can perform is a chosen plaintext attack, injecting traffic and observing the resulting ciphertext. Due to the nature of the XOR operation, the adversary can recover the one-time pad used to encrypt a packet's IP address if it knows the original address. However, since SPINE generates different nonces and therefore different one-time pads *for each packet*, the ability to do so will not assist in recovering future encrypted IP addresses. The fact that SPINE generates a fresh nonce and one-time pad for each packet and encrypts TCP sequence and acknowledgment numbers also allows it to achieve a strong privacy property that we call *flow-packet unlinkability*: packets belonging to the same flow will have different (encrypted) IP addresses and meaningless (encrypted) TCP sequence and acknowledgment numbers, which prevents an adversary from associating packets with flows.

3.2 Consistency Across Key Rotation

SPINE updates the secret keys being used by SPINE routers every t seconds. Key rotation reduces the damage caused by compromised secret keys. Even if an attacker somehow manages to obtain the current keys, such knowledge eventually becomes useless after (at most) t seconds. Key rotation also ensures the freshness of one-time pads: even if there are collisions in nonces, the keyed hash function produces different one-time pads under different keys. Another benefit of key rotation is to give participating ASes the option of not being able to retroactively decrypt IP addresses. Without keeping a record of the keys used over time, the ASes will be unable to decrypt older traffic. This is useful because if a government entity were to order the AS to decrypt packets, the AS would be unable to decrypt any packets sent before the current set of keys was adopted (i.e., the AS would have plausible deniability).

One potential issue caused by key rotation is inconsistent keys during encryption and decryption. For instance, referring back to Figure 1, the shared secret keys may be updated when the packets from border router R_1 are still in transit; without

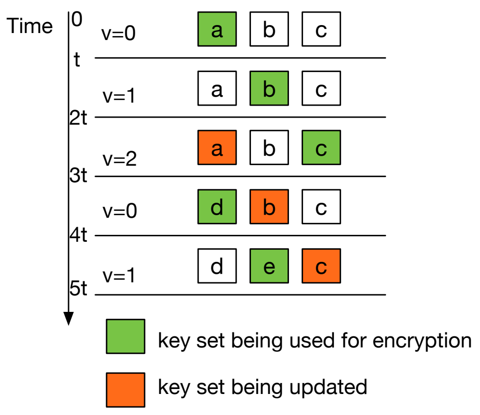

Figure 2: SPINE key and version rotation.

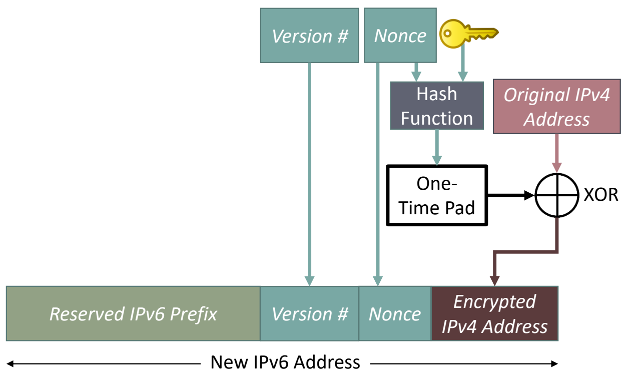

Figure 3: Encrypted IPv4 Address Encoded as IPv6 Address.

recording the keys used before, R_2 cannot decrypt the packets that are encrypted with outdated keys.

Recall that SPINE uses one key to encrypt IP addresses and another key to encrypt TCP fields (when present); we call these two keys together a *key set*. Because we want participating ASes to achieve plausible deniability, SPINE does not remember all outdated key sets. Instead, to address the inconsistency issue described above, SPINE remembers the last key set used for a certain period of time. More specifically, SPINE maintains three key sets (the previous set, the current set, and the next set) at any given time. Assuming that there are three slots $slot_i (i = 0, 1, 2)$ for storing key sets, SPINE rotates a slot index v, from 0 to 2, every t seconds, and the key set in $slot_v$ is used for encryption. A key set can stay in a slot for at most $3t$ seconds; after $3t$ seconds, the key set is updated by SPINE. An example is illustrated in Figure 2.

The encrypting party will send the version number (i.e., the slot index) of the key set used for encryption in packet headers. Based on the version number, the decrypting party retrieves the appropriate key set for decryption. Because the previous key set is remembered, the decrypting party can decrypt the packets encrypted with the previous key set. We assume that we pick a long enough time t such that there are no more packets encrypted with a given key set in transit in the network before that key set is discarded.

We assume that all routers running the SPINE protocol are coordinated from a logically centralized controller, which performs key rotation, updates version numbers, and passes updated keys and versions as arguments to all routers running the SPINE protocol.

3.3 Encoding and Routing Using IPv6

Encoding information in SPINE faces three challenges:

1. We need to send the nonce, version number, and encrypted IP addresses along with the packet.
2. We need a way to discern which traffic has been encrypted and thus needs to be decrypted when it exits the untrusted network entities.
3. We need to ensure that packets are still forwarded successfully because we are encrypting IP addresses, and routing depends on IP addresses.

To address these three issues, SPINE transforms IPv4 headers into IPv6 headers when packets enter the untrusted entity (e.g., when packets travel from R_1 to the untrusted entity in Figure 1) and transforms the IPv6 headers back into IPv4 headers when the packets exit the untrusted entity (e.g., when the same packets exit the untrusted entity at R_2), which is similar to NAT64 [12]. These transformations mostly preserve the IPv4 header fields that have corresponding IPv6 header fields (e.g., TTL maps to hop limit, etc.). The primary exception is the IP address field because SPINE uses the IPv6 source and destination address fields to solve the three problems outlined above. The first 64 bits of the new IPv6 destination address contain a reserved /64 prefix d_r owned by the receiving trusted entity, the next 32 bits contain the version number and nonce, and the last 32 bits contain the encrypted IPv4 address (see Figure 3); a similar scheme is used to create the new IPv6 source address. The original TCP sequence and acknowledgment numbers are simply replaced by their encrypted counterparts.

This method of creating IPv6 addresses addresses the first challenge. The reserved prefix d_r addresses the second and third challenges. SPINE assumes that the two trusted entities coordinate on the /64 IPv6 prefix d_r beforehand. The receiving trusted entity announces this prefix but does not offer any services at these addresses. It can then assume that any packets arriving with addresses with the d_r prefix are part of the SPINE scheme. This allows trusted entities to determine which traffic needs to be decrypted, solving the second problem. Regarding the third problem, we note that routing tables are populated by advertising routes to prefixes. Since the receiving trusted entity owns d_r, packets with the d_r prefix will be correctly forwarded to it. IPv6 prefixes are typically 64 bits, so our requirement that d_r is a /64 prefix is reasonable.

Using IPv6 addresses therefore provides several benefits:

- The length of IPv6 headers gives us bits to use for metadata, which facilitates SPINE's encryption.
- The standardized lengths of the prefix and host portions of IPv6 addresses allow us to ensure that routing is successful even when we encrypt IPv4 addresses.
- Because there are so many prefixes in the IPv6 space, having a reserved d_r prefix is a feasible option, which makes the differentiation between encrypted and non-encrypted headers very easy.

The d_r prefixes reveal some information about the source and destination IP addresses, but if we must rely on the untrusted entity's routers to carry network traffic, then we cannot avoid revealing this information. Because the untrusted entity is directly receiving packets from one trusted entity and forwarding these packets to the other trusted entity, there is no way for packets to be forwarded successfully without the adversary knowing this information.

4 Prototype Implementation

After a brief overview of the Protocol Independent Switch Architecture (PISA), we describe how we implemented important aspects of SPINE (e.g., nonces and hashes) in P4, and then discuss the resource requirements for the PISA switch.

4.1 Background: PISA Switch Architecture

The PISA pipeline consists of a programmable parser, a series of match-action tables, and a programmable deparser. The parser allows the programmer to define headers that are extracted when packets arrive at the switch. The match-action tables, which generally perform matching on some part of the parsed header, are used to choose actions for a packet (e.g., drop it, manipulate its headers, etc.); these tables can also specify arguments to be passed into these actions. The deparser emits the final packet. If we can implement SPINE such that it compiles to the PISA switch hardware target, then, by construction, SPINE will operate at switch hardware rates. In addition to these resource constraints, we are also limited by the allowed functionality in P4 (e.g., no loops).

We implement SPINE in $P4_{16}$, which is the 2016 updated revision of the P4 language [15]. We also use a software switch called `simple_switch`, which is built to execute $P4_{16}$ programs. Figure 4 shows a flowchart of the SPINE P4 packet processing pipeline for the border router in Trusted Entity #1 in Figure 1. The keys and reserved d_r prefixes are stored in Tables 2 and 5 and are passed as arguments to actions when these tables are called.

As discussed above, SPINE assumes the existence of a central controller that writes all values in the match-action tables. The central controller in the prototype implementation is a Python program that establishes a gRPC connection to each switch. Writing the tables at runtime means we can modify tables after installation to insert new keys.

We use PISA switches and P4 for the implementation because they ensure that SPINE will run at switch hardware rates if it fits in switch memory. PISA switches are also currently being deployed in edge networks. Given that the trusted entities (Figure 1) can easily be modeled as edge networks, this means that SPINE will be feasibly deployable.

4.2 Evaluation

Developing mechanisms to efficiently generate nonces and compute hashes posed several challenges due to the computational and memory limitations of PISA switches. To generate nonces, we use P4's `random()` function.

Finding a hash function that can be implemented in P4 is more challenging, especially given P4's memory constraints and the fact that hash functions are often computationally intensive. Our implementation relies on SipHash [1], a high-speed pseudorandom function that is optimized to perform well on short messages, requires just four 64-bit variables, and only involves XOR and addition operations. These all make SipHash amenable to a P4 implementation. SipHash takes as input a 128-bit secret key and a message (in this case, the nonce) and outputs a 64-bit hash. We split the 64-bit hash into two 32-bit one-time pads to encrypt either 32-bit IPv4 addresses or 32-bit TCP sequence and acknowledgment numbers. Keys and versions are passed as arguments to routers running the SPINE protocol when the match-action Tables 2 and 5 in Figure 4 are called. The central controller initializes six keys (3 key sets containing 2 keys each) as arguments in the match-action tables. To update these keys, the central controller simply changes the arguments in these match-action tables.

We have released a prototype implementation of SPINE[1], which extends the public P4 tutorials [14], and tested its operation and correctness using the Mininet network emulator.

4.3 Resource Requirements

Our prototype allows us to evaluate the resource requirements of SPINE. If the implementation can fit on a PISA switch, then it will run at switch hardware rates on this switch. The P4 implementation uses five tables, as shown in Figure 4. Table 3 is a normal IPv4 forwarding table that all v4 switches must have, and Tables 1, 2, 4, and 5 are additional tables used to implement SPINE.

To discuss the resource requirements of our implementation, we consider a scenario where multiple trusted entities are communicating with each other through one or more untrusted entities. In this situation, the number of lines in Table 1 for a border router in one of the trusted entities is at least equal to the number of reserved d_r prefixes that the other trusted entities advertise. Each of these lines consists of a 128-bit number representing a single d_r and a 1-byte number representing the length of the prefix. Note that Table 1 also contains lines that allow for standard IPv6 routing, but because any

[1]https://github.com/SPINE-P4/spine-code

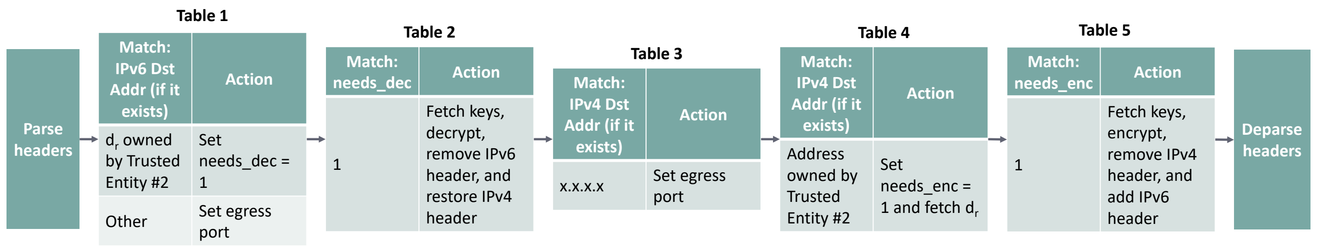
Parse headers
Table 1
Match: IPv6 Dst Addr (if it exists)
Action
d_r owned by Trusted Entity #2
Set needs_dec = 1
Other
Set egress port
Table 2
Match: needs_dec
Action
1
Fetch keys, decrypt, remove IPv6 header, and restore IPv4 header
Table 3
Match: IPv4 Dst Addr (if it exists)
Action
x.x.x.x
Set egress port
Table 4
Match: IPv4 Dst Addr (if it exists)
Action
Address owned by Trusted Entity #2
Set needs_enc = 1 and fetch d_r
Table 5
Match: needs_enc
Action
1
Fetch keys, encrypt, remove IPv4 header, and add IPv6 header
Deparse headers

Figure 4: P4 Implementation on Trusted Entity #1 (ingress) border router.

switch that enables IPv6 routing would require such lines, we do not count them toward our resource requirements. The number of lines in Table 4 is equal to the number of prefixes owned by the other participating trusted entities. Each line in Table 4 consists of one 32-bit number representing an IPv4 prefix, a 1-byte number representing the length of the prefix, and a 128-bit number representing the corresponding reserved IPv6 prefix d_r. Table 2 contains one line of 6 128-bit keys (96 bytes), and Table 5 contains the same keys and the current version number v (approximately 97 bytes). Note that we have separate tables for fetching keys so that we only have to store the keys once instead of on every line in Tables 1 and 4 (this also makes updating keys easier). Thus, if there are a total of R reserved IPv6 prefixes advertised by the participating trusted entities and a total of P IPv4 prefixes owned by the trusted entities, then the tables in a single border router will take up approximately $17R+96+21P+97=17R+21P+193$ extra bytes of memory in addition to the normal IPv4 and IPv6 forwarding table. According to CIDR Report [7], the largest numbers of IPv4 prefixes and IPv6 prefixes announced by an AS are 5,759 and 1,383, respectively. Storing these prefixes only requires 144,643 bytes or 0.14 MB extra memory.

5 Related Work

One main area of related work is network layer anonymity systems. In such systems, anonymous circuits are first established by senders and participating routers, and then traffic is forwarded through these circuits. LAP [10], an early example of a network layer anonymity system, relaxes privacy guarantees to achieve higher speeds. Dovetail [19] achieves similar privacy guarantees but uses a intermediate node to construct traffic paths and prevent leaks that are possible in LAP. HORNET [4] is a high-speed scalable onion routing system that achieves better privacy guarantees than LAP and Dovetail. PHI [6] offers better privacy guarantees than LAP and Dovetail and better performance than LAP, Dovetail, and HORNET. TARANET [5] improves on HORNET by using a new method of traffic shaping to thwart traffic analysis attacks. The main difference between SPINE and network layer anonymity systems is that the latter assumes that all routers on the path participate in the protocol. This assumption is a high barrier to deployment because requiring that all ASes on the path, especially potential adversaries, implement such a protocol is not practical. In contrast, SPINE does not assume that routers in adversarial ASes implement any protocols other than standard packet forwarding. Furthermore, network layer anonymity systems often require hosts to initiate anonymous connections, whereas SPINE leverages the security and technical knowledge of AS owners to protect (often uneducated) end users without requiring them to take action.

The most similar work to ours is the Address Hiding Protocol (AHP) [18], which, like SPINE, enlists ISPs and ASes to protect end user privacy. AHP assigns a different random address from an ISP's address block to each traffic flow started by a client of the ISP. AHP therefore uses the same IP address for each flow, unlike SPINE. This method of obfuscation does not protect against the traffic-analysis attacks permitted in the threat model. In addition, shorter prefixes imply a larger set of possible addresses from which to randomly choose an IP address for a client flow. This means that AHP is most effective in large ISPs that own short prefixes. In contrast, the standardized prefix lengths of IPv6 addresses mean that SPINE is equally effective for ISPs and ASes of all sizes.

6 Conclusion

We have presented SPINE, which allows two trusted network entities to communicate through untrusted network entities without revealing source and destination IP addresses of traffic. SPINE encrypts IPv4 addresses and relevant TCP fields using efficient one-time pad encryption and stores the encrypted IPv4 addresses in IPv6 headers that are added to packets when they enter the untrusted network entities and are removed when they exit. In contrast to previous solutions that face high deployment barriers, SPINE needs to be implemented only by the participating trusted network entities and requires no action from end users. We presented a P4 implementation of SPINE for PISA switches, which can run at switch hardware rates and demonstrated that the resource requirements of this implementation are feasible for a practical, real-world deployment.

Acknowledgments: We thank Prateek Mittal and the anonymous reviewers for their comments. This research was supported by DARPA under Dispersed Computing HR0011-17-C-0047 and NSF TWC Award 1602399.

References

[1] J.-P. Aumasson and D. J. Bernstein. Siphash: A Fast Short-Input PRF. In S. Galbraith and M. Nandi, editors, *Progress in Cryptology - INDOCRYPT*, pages 489–508, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, July 2014.

[3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM*, 2013.

[4] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. HORNET: High-speed Onion Routing at the Network Layer. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1441–1454, New York, NY, USA, 2015. ACM.

[5] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso. TARANET: Traffic-Analysis Resistant Anonymity at the NETwork layer. *CoRR*, abs/1802.08415, 2018.

[6] C. Chen and A. Perrig. PHI: Path-Hidden Lightweight Anonymity Protocol at Network Layer. In *Privacy Enhancing Technologies*, pages 100–117, 2017.

[7] CIDR report. https://www.cidr-report.org/as2.0/.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[9] A. Edmundson, R. Ensafi, N. Feamster, and J. Rexford. Nation-State Hegemony in Internet Routing. In *ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 17:1–17:11. ACM, 2018.

[10] H. Hsiao, T. H. Kim, A. Perrig, A. Yamada, S. C. Nelson, M. Gruteser, and W. Meng. LAP: Lightweight Anonymity and Privacy. In *IEEE Symposium on Security and Privacy*, pages 506–520, May 2012.

[11] A. Mari. High Expectations for Brazil Undersea Cable to Europe. *ZDNet*. https://www.zdnet.com/article/high-expectations-for-brazil-undersea-cable-to-europe/.

[12] NAT64 technology: Connecting IPv6 and IPv4 networks. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/enterprise-ipv6-solution/white_paper_c11-676278.html.

[13] One-Time Pad (OTP). *Crypto Museum (archived)*. https://web.archive.org/web/20140314175211/http://www.cryptomuseum.com/crypto/otp.htm.

[14] P4 Tutorial. https://github.com/p4lang/tutorials.

[15] P4_16 Language Specification. https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html, May 22 2017.

[16] S. Patil and N. Borisov. What can you learn from an IP? In *Applied Networking Research Workshop*, pages 45–51. ACM, 2019.

[17] The World's Fastest & Most Programmable Networks (whitepaper). https://barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/.

[18] B. Raghavan, T. Kohno, A. C. Snoeren, and D. Wetherall. Enlisting ISPs to Improve Online Privacy: IP Address Mixing by Default. In I. Goldberg and M. J. Atallah, editors, *Privacy Enhancing Technologies*, pages 143–163, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[19] J. Sankey and M. Wright. Dovetail: Stronger Anonymity in Next-Generation Internet Routing. In E. De Cristofaro and S. J. Murdoch, editors, *Privacy Enhancing Technologies*, pages 283–303, 2014.