Programmable In-Network Obfuscation of DNS Traffic

Liang Wang, Hyojoon Kim, Prateek Mittal, Jennifer Rexford
Princeton University

ABSTRACT

In conventional DNS, or Do53, requests and responses are sent in cleartext. Thus, DNS recursive resolvers or any on-path adversaries can access privacy-sensitive information. To address this issue, several encryption-based approaches (e.g., DNS-over-HTTPS) and proxy-based approaches (e.g., Oblivious DNS) were proposed. However, encryption-based approaches put too much trust in recursive resolvers. Proxy-based approaches can help hide the client's identity, but sets a higher deployment barrier while also introducing noticeable performance overhead. We propose PINOT, a packet-header obfuscation system that runs entirely in the data plane of a programmable network switch, which provides a lightweight, low-deployment-barrier anonymization service for clients sending and receiving DNS packets. PINOT does not require any modification to the DNS protocol or additional client software installation or proxy setup. Yet, it can also be combined with existing approaches to provide stronger privacy guarantees. We implement a PINOT prototype on a commodity switch, deploy it in a campus network, and present results on protecting user identity against public DNS services.

1 INTRODUCTION

The Domain Name System (DNS), responsible for translating domains to IP addresses, is a critical component of the Internet infrastructure. A user's DNS request is usually handled by a DNS recursive resolver, which performs DNS lookups on behalf of the user and returns the resolved IPs. DNS recursive resolvers are typically operated by the user's Internet service provider (ISP) or third-party services (e.g., Google and Cloudflare). In conventional DNS (Do53), requests and responses are in cleartext, so DNS recursive resolvers or on-path adversaries have access to the user's IP address and the domain visited. Such information can be used for inferring users' browsing behaviors, fingerprinting and tracking users, or even deanonymizing Tor users [19, 22, 42].

Encryption-based approaches have been proposed to protect user privacy in DNS. For instance, DNSCrypt [16], DNS-over-HTTPS (DoH) [23], and DNS-over-TLS (DoT) [25] send encrypted DNS requests to prevent eavesdroppers from seeing the queries. However, encryption-based approaches put trust in the recursive resolvers, as the recursive resolvers see the user's real IP address and the associated queries, making the recursive resolvers a single point of privacy failure.

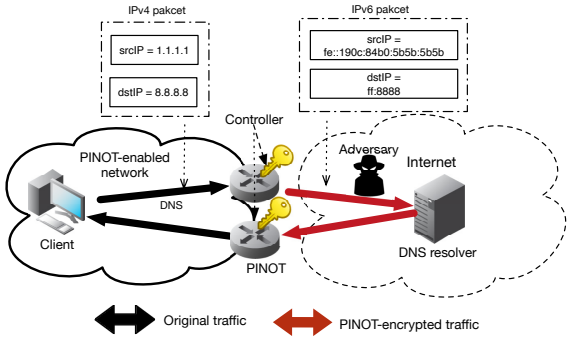

Figure 1: PINOT setup.

Proxy-based approaches try to address this issue, leveraging on-path proxies between a DNS client and a resolver to decouple client IP addresses from DNS queries. Such examples are DNS over Tor, Anonymized DNSCrypt [14], Oblivious DNS (ODNS) [37], and Oblivious DoH (ODoH) [39]. Proxy-based approaches are complementary to encryption-based approaches and their combination offers better privacy guarantees. However, this is at the expense of performance degradation (i.e., higher DNS resolution latency) and load on the proxies. The proxy-based approaches require modifications to the DNS protocol and client-side software, further raising deployment barriers.

According to Cloudflare, 92% of the daily requests (254 Billion/day) received by a popular large recursive DNS resolver still use Do53 (i.e., conventional DNS) [39]. Considering that Do53 will continue to have a large user base for the foreseeable future, it is important to protect user privacy in Do53. This begs the question: *Can we provide a lightweight, readily-deployable client anonymization solution for DNS traffic without any modification to the DNS protocol and infrastructure?*

1.1 IP address obfuscation in data plane

Hiding the original client IP address from recursive resolvers and untrusted networks is essential for preserving user privacy in DNS traffic. Yet, providing such a capability normally requires traversing trusted proxy nodes in the Internet, which

typically involves installing additional client software. In our work, we leverage recent advances in programmable switch hardware to develop a first-of-its-kind, **network-based** approach for enhancing DNS privacy. We present PINOT (Programmable In-Network Obfuscation of Traffic), a lightweight packet-header obfuscation system that runs entirely in the data plane of a programmable network switch. Serving as an in-network transparent proxy, PINOT aims at preventing an adversary from associating a DNS request with the IP address of its originating client. PINOT does not require any modification to the DNS protocol, nor does it require any additional client software installation. We design PINOT as a privacy primitive for obfuscating the association between client IP addresses and DNS requests, and demonstrate that it is a useful building block for bootstrapping advanced DNS privacy applications when being used in combination with other privacy-enhancing solutions. In this preliminary work, we focus on protecting client IPv4 addresses against particular public DNS services, and discuss protecting IPv6 addresses in §3.3.

PINOT runs at the border of a trusted network (e.g., an enterprise network) and encrypts the users' IPv4 addresses in DNS traffic to random IPv6 addresses owned by the network before packets leave the network, as shown in Figure 1. PINOT obfuscates each packet individually with a probabilistic encryption scheme so that successive requests from the same user have different encrypted IP addresses. Programmable switch hardware has limited memory and processing resources, posing challenges for implementing cryptographic algorithms that are commonly used in privacy applications. Nevertheless, we use a secure and efficient cipher built atop iterated Even-Mansour (EM) [2], which can perform encryption and decryption in a manner that is compatible with a single pass through the packet-processing pipeline of a hardware switch. Thus, PINOT can encrypt/decrypt IP addresses at hardware switch rates (e.g., up to 12.8 Tbps on a state-of-the-art Intel Tofino switch) [26].

Leveraging the growing deployment of IPv6 in the core of the Internet, PINOT converts an IPv4 packet to an IPv6 packet that embeds the encrypted IPv4 address in the IPv6 source address (similar to NAT46 [11]) to (1) carry encryption-related state and (2) ensure response packets can be forwarded back normally. In contrast to stateful NATs [10], PINOT is *stateless* and does decryption entirely based on the information in IPv6 addresses, which makes it scalable and also allows it to handle asymmetric routing. With a software controller distributing the per-AS secret keys, an AS can deploy PINOT at multiple border points of the trusted network; outgoing DNS traffic can go through any egress points while return traffic can come into any ingress point.

After encryption, the source IP address in a packet becomes meaningless to an adversary, who only knows which Autonomous System (AS) initiated the packet but cannot pinpoint the specific host that sent the packet, nor associate multiple packets as coming from the same host. Our security analysis demonstrates that PINOT is secure against a realistic adversary under practical constraints.

	Query encryption	**IP hiding**	**Transparency**	**Proxy overhead**
Do53	No	No	-	-
Encryption (DoH, DoT, DNSCrypt, etc.)	Yes	No	No	-
Proxy (ODNS, ODoH, DNS over Tor, etc.)	Yes	Yes	No	High
PINOT + Do53	**No**	**Yes**	**Yes**	**Low**
PINOT + Encryption	**Yes**	**Yes**	**Yes***	**Low**

Table 1: A comparison of PINOT and its variants with different approaches. *Assuming an encryption-based solution has already been deployed, using PINOT to achieve IP obfuscation does not require modifying the existing client/server software.

Real-world deployment. We implement a prototype of PINOT on an Intel Tofino switch [26] and deploy it in a campus network to forward DNS traffic. We have released the source code of PINOT to facilitate future research [33]. We show PINOT is feasible as it can correctly encrypt, decrypt, and forward DNS traffic. While previous works have implemented cryptographic algorithms on programmable data planes using CPUs, SmartNICs, or NetFPGAs [20, 21, 38], to the best of our knowledge, we are the first to implement a working secure and efficient encryption scheme that runs at hardware switch rate within programmable switch ASICs.

1.2 PINOT for DNS privacy

PINOT + Do53. PINOT can be deployed at the border of a network to encrypt the source IP address in each outgoing Do53 DNS request individually. We assume each DNS request fits into a single UDP packet. [1] Based on the DNS traffic collected in two large enterprise networks, DNS requests are usually less than 512 bytes [1]. A DNS response can be split across multiple packets. This would not affect the decryption in PINOT, because PINOT is stateless and all the information (other than the secret key) required for decryption is stored in the destination IPv6 address.

Unlike encryption-based and proxy-based DNS privacy solutions, PINOT is an in-network solution and requires no participation of end-users (DNS clients and resolvers). It also makes no modifications to the existing DNS infrastructure and

[1] This assumption can be relaxed, see §A.3.

protocol, and thus is completely *transparent*. Once deployed, PINOT can provide *ubiquitous protection* to all of the users in the trusted network. There is no extra effort needed from the users. Table 1 summarizes how PINOT compares to other approaches. Importantly, leveraging programmable switches, PINOT adds negligible latency to DNS traffic.

PINOT is a standalone solution. Yet, PINOT is *compatible* with certain encryption-based approaches (PINOT + Encryption in Table 1), and can offer better DNS privacy with low proxy overhead and throughput degradation when used together. Adding PINOT to existing encryption-based approaches does not require any additional change to the DNS clients and resolvers.

PINOT + DNSCrypt. PINOT decouples IP (header) protection from DNS request encryption, and thus enables a more flexible choice of the request protection scheme. For instance, one can use DNSCrypt [16] to encrypt the DNS requests to achieve confidentiality. In this case, the trusted network that deploys PINOT would not be able to see the plaintext DNS requests, while the resolver cannot see the real client IP addresses. The resulting system achieves similar privacy guarantees as Anonymized DNSCrypt [14]. However, PINOT can encrypt packets at line rate with almost zero latency overhead, avoiding the performance bottleneck of proxy-based approaches.

PINOT + DoH/DoT. With a few modifications PINOT can also work with connection-oriented protocols such as DoH and DoT. In this case, PINOT needs to maintain per-connection state and encrypt IP addresses on a per-connection basis, which works in a similar way as a NAT. We leave extending PINOT to support connection-oriented protocols as future work.

The above cases demonstrate that PINOT can enhance privacy of Do53 and several encryption-based DNS without any changes or effort from other parties, other than the AS/network that has an interest in boosting user privacy. ISPs or cloud providers may also want to participate in PINOT because of financial incentives, e.g., a cloud provider can deploy PINOT as a privacy-enhancing service to attract more customers.

2 PINOT PROBLEM DEFINITION

We study how to design a readily-deployable, in-network privacy-enhancing system for DNS traffic with minimal performance overhead. As the first step towards a comprehensive in-network anonymity system, we consider a lightweight notion of anonymity, in which the AS where PINOT is deployed is trusted and can observe both ends of a communication session. [2] We only consider protecting the user's IP address against widely-used DNS public services. In this section, we elaborate on design goals, threat model, and hardware resource constraints.

2.1 Design goals

We seek to design a system that achieves the following privacy properties:

Sender anonymity. With sender anonymity, an adversary cannot discover the identity (IP address) of the client (sender). We do not try to hide the server's identity or the Autonomous System (AS) of the client.

Packet unlinkability. We define packet unlinkability for connectionless protocols as: given packets sent from a set of clients, the adversary cannot determine whether the packets are associated with the same client based on observed IP addresses. This property helps protect users against traffic-analysis attacks or user tracking. *Associating packets to clients using non-IP information (packet size, timing, etc.) is beyond the scope of this paper.*

Additionally, we want to achieve two operational goals:

Low deployment barriers. The solution should be readily deployable without modifications to existing Internet infrastructure and protocols, or running special client-side software (i.e., no involvement of end-users). In addition, the solution should be able to provide ubiquitous privacy protection for a set of users.

Low performance overhead. We want our solution to process network traffic at hardware switch rates. Therefore, we need to minimize the overhead introduced by cryptographic operations, and keep as little per-packet/per-flow state as possible (or, better yet, no state at all).

2.2 Threat model and assumptions

As shown in Figure 1, we assume an *unmodified* client communicates with a server through a trusted entity (an enterprise network or an ISP). The trusted entity and the server should have both IPv4 and IPv6 connectivity. The goal of an adversary is to recover the actual client IP addresses of network traffic it observes, given the contents of the packets that it can see.

We consider two types of attackers: passive and active. The passive adversary could be the remote server, or any network element between the trusted network and the server. The adversary may be an AS; it could also be any intermediate network point such as an Internet exchange point (IXP) or even simply a link. An active adversary may control a few hosts in the trusted network, and is able to send packets with

[2] The combination of PINOT and encryption based methods will prevent the deploying AS from learning client DNS queries.

arbitrary spoofed source IP addresses. We assume, however, that an active adversary cannot observe other users' traffic in the trusted network.

Finally, we do not yet consider implementation-specific attacks, such as bugs in the implementation and bias in the random number generators offered by hardware switches.

2.3 Hardware switch resource constraints

High-speed programmable switches can facilitate achieving the performance goal in §2.1. Programmable switches deployed at the border of the trusted network can process terabytes of traffic per second. Nevertheless, to build a system with the desired privacy properties, we need to work carefully within the hardware resource constraints.

In a programmable switch, the packet-processing pipeline is divided into multiple *stages*. Each stage only allows a limited number of table lookups, and mathematical and logical operations. A program running in the data plane can process traffic at line rate only if it can "fit" into the switch ASIC—that is, if the program only requires the packet to go through the pipeline once. Given the limited stages in commodity programmable switches, fitting standard cryptographic algorithms into the switch, if possible, is extremely challenging (see §6 for more discussions).

3 PINOT DESIGN

PINOT consists of a software controller for key management, and a data-plane program that performs IP address encryption/decryption at hardware switch rates using a lightweight secure cipher [2]. We use the IPv6 address encoding technique to avoid maintaining any per-flow or per-packet state, similar to NAT46 [11]. With the controller distributing the per-network encryption keys, PINOT can be deployed at multiple border points of the trusted network and naturally handles asymmetric routing. Next, we discuss the encryption scheme and IPv6 encoding.

3.1 Efficient encryption in the data plane

Standard encryption algorithms such as AES are too complex to implement in a single pass through the switch ASIC. Performing multiple passes over the packets would cause significant performance degradation, making line-rate encryption infeasible. Therefore, we look for a lightweight and secure cipher that can fit into switch ASICs. After exploring various options, we found that the two-round Even-Mansour (2EM) scheme [2] satisfies our requirements. 2EM can be implemented using table lookups and XORs, avoiding complex cryptographic computations such as hashing in the data plane. With careful code optimization, 2EM can encrypt a packet in a single pass through the switch packet-processing pipeline, yet is secure against a computationally bounded adversary in practice.

Our cipher encrypts a n-bit message M by computing:

$$E(M) = P_2(P_1(M \oplus k_0) \oplus k_1) \oplus k_2 \quad (1)$$

where k_0, k_1, and k_2 are n-bit *independent* encryption keys to thwart attacks that exploit key relation, and P_1 and P_2 are *independent* permutations over n-bit strings, which can be implemented as substitution-permutation networks (SPN) [41]. This construction has been proven to be secure up to $2^{\frac{2n}{3}}$ queries under adaptive chosen-plaintext and ciphertext adversaries [8, 27]. For encrypting 32-bit IPv4 addresses ($n = 32$), 2EM is only secure against 2.6 million queries, i.e., an adversary can recover keys or plaintexts with high probability after knowing 2.6 M plaintext-ciphertext pairs using efficient attacks [8, 30]. To improve the security of PINOT, we adopt two approaches:

• *Random padding to increase message size*: We append a l-bit random string to an IP address to extend the length of encryption input. The use of random padding improves the security of the encryption as n becomes larger ($l + 32$). For $l = 32$, our cipher is secure against about 7 trillion queries. Random padding also makes the encryption non-deterministic, i.e., encrypting a given IP addresses multiple times will produce different ciphertexts. This is desirable for achieving packet unlinkability.

• *Key rotation to limit the number of encryptions under given keys*: We update the *key set* (i.e., the three encryption keys k_0, k_1, and k_2) being used for encryption every t seconds. Key rotation limits the number of plaintext-ciphertext pairs an adversary can collect for given keys to reduce the attack success probability, and minimizes the damage caused by compromised encryption keys, as keys expire after at most t seconds.[3] One potential issue caused by key rotation is inconsistent keys during encryption and decryption, i.e., the key set may be updated when the packets from the server are still in transit. To address this issue, we maintain three versions of key sets, and rotate the key sets using the algorithm proposed in SPINE [13]. The key version number is encoded in the encrypted client address (§3.2).

Practical attacks are hard under realistic resource constraints. A realistic adversary can only perform a reasonable amount of computation (i.e., computationally bounded) under limited memory resources. The best known practical attacks against 2EM, which are chosen-plaintext attacks, require more than 2^{89} bits of memory for $n = 64$ even if the adversary can send more than 1 trillion queries per second [15]. That would generate approximately 2.4 Pbps DNS traffic when assuming the average IPv4 packet size is 300 bytes [4]. This is infeasible in practice. In addition, generating a large volume

[3] We can also rotate the permutations.

of traffic from a few hosts, if possible, can easily be flagged as DDoS attacks. See Appendix §A.1 for a detailed security analysis.

The adversary may also target a specific IP address, spoof this IP address, and collect all the possible encrypted IPv6 addresses to build a cipertext table in advance. If it finds a packet's encrypted source IPv6 address in the table, the adversary knows the packet was sent from the target IP address. However, to have a high success rate, such attacks require the adversary to generate about 10 Tb traffic from a single IP address to build the cipertext table for a given key set, which can also easily be detected.

3.2 Translation from IPv4 to IPv6

Encrypting a 32-bit IPv4 address will produce a ciphertext of $32 + l$ bits (l is the length of the random padding), which cannot be used as a valid IPv4 address for routing. The whole ciphertext also needs to be stored somewhere for decryption. To ensure return traffic can be correctly routed back based on the encrypted IP addresses and to avoid maintaining any state in the switches, we transform an IPv4 packet into an IPv6 packet when the packet leaves the trusted network. The IPv6 packet encodes the encrypted IPv4 address in its IPv6 source address field, as shown in Figure 2. The highest d bits of a transformed IPv6 source address are the IPv6 network prefix reserved by the trusted network, the lowest $32 + l$ bits contain the encrypted IPv4 source address, and the remaining bits are used for storing encryption metadata such as the key set version number. The values in the IPv4 header fields that have corresponding IPv6 header fields are preserved.

The IPv4 destination addresses are replaced with their IPv6 counterparts. Recall that we only consider services with both IPv4 and IPv6 addresses. Therefore, we can perform DNS lookups in advance to get the IPv4 and IPv6 addresses of public servers of interest from their DNS A and AAAA records, respectively, and store the IPv4/IPv6 address pairs in a lookup table in the switch data plane for later use.

We use the key version number in the IPv6 destination address of a return (IPv6) packet to locate the key set for decryption. The return packet is converted back to an IPv4 packet and forwarded based on the decrypted address.

3.3 PINOT for IPv6 networks

We can also use PINOT to protect certain IPv6 networks. For instance, suppose the trusted entity has a /64 IPv6 network, and reserves a /96 network for PINOT. We call the 64 to 96 bits in an IPv6 source address *subnet ID*, which is static in this case. We can use PINOT to encrypt the lowest 32 bits of an IPv6 source address with up to 30 bits of random padding, and use the subnet ID to carry the encryption meta. During decryption, PINOT replaces the subnet ID part of an IPv6 address with the original, static subnet ID.

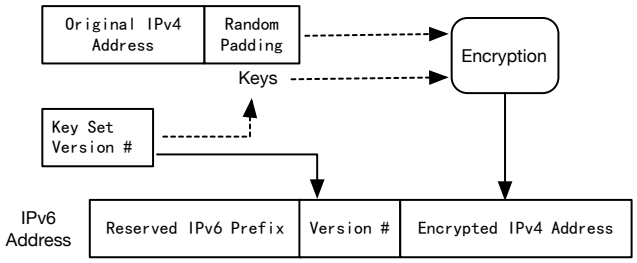

Figure 2: PINOT IPv6 source address encoding.

4 IMPLEMENTATION ON COMMODITY SWITCH

Key rotation and distribution. PINOT consists of a software controller for key distribution and rotation, and a P4 [3] data-plane program for IP address encryption. The controller, which can run on a dedicated host or in the control plane of a programmable switch, generates three 64-bit encryption keys using the Python urandom function. It uses grpc to communicate with the data plane to update the keys and the key version number. In our evaluation, the keys are updated every five seconds. The two-bit version number is stored along with the port forwarding (i.e., switch ingress port to egress port) information in a forwarding table.

Address mapping. For IPv6/IPv4 packet transformation, PINOT also maintains two address mapping tables (*IP4to6* and *IP6to4*) that store the corresponding IPv6 address of an IPv4 address, and vice versa, for the DNS resolvers.

2EM encryption. PINOT generates random paddings via the Random external function in the Tofino programmable switch, and uses substitution-permutation networks for permutation [41]. To permute a 64-bit input, PINOT first performs substitution using 8-bit substitution boxes (S-boxes) to substitute every byte of the input with another byte, and then applies a 64-bit straight permutation box (P-box) to shuffle the bits of the S-box output. We currently use the static S-box in the AES standard and randomly shuffle the bits in the P-box; we plan to generate S-boxes and P-boxes dynamically and update them periodically in the future.

We implement two different PINOT prototypes: a 56-bit version and a 64-bit version that use 24-bit and 32-bit random paddings (i.e., $l = 24$ and $l = 32$), respectively. We use the 56-bit version for the real-world deployment because we have a /64 IPv6 network allocated; with the 64-bit version, there is no space left in an IPv6 address for the two-bit version number. Figure 3 shows an example of IPv4 source address encryption in the 56-bit PINOT. Even with the 64-bit PINOT,

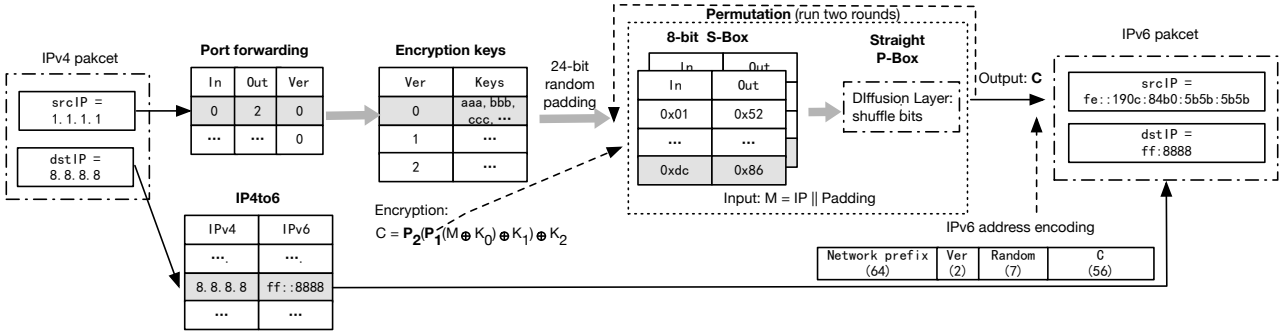

Figure 3: IPv4 source address encryption in the data plane in 56-bit PINOT.

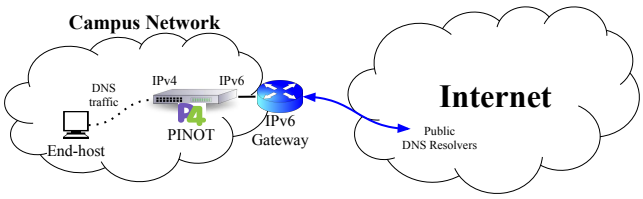

Figure 4: PINOT deployment in a campus network.

the S-box or inverse S-box tables only take up about 16 KB of memory, and the extra memory used for storing encryption keys and P-boxes are negligible. Such low overhead allows PINOT to store additional encryption keys to encrypt more fields in the packet header. See Appendix §A.2. We have released the source code of PINOT [33].

5 DEPLOYMENT AND EVALUATION

Wide-area testbed. We test PINOT on Do53 traffic in our evaluation. Figure 4 shows the PINOT deployment in our network that connects to the wider Internet. The end-host IPv4 client device is a Linux server with two Intel Xeon E5530 2.4GHz CPUs and 16GB of memory. The PINOT switch sits between our end-host device and the trusted network's border gateway. The switch is a Wedge 100BF-32X switch with a Tofino programmable chip [29], and loads the PINOT `P4_16` program. The switch acts as the IPv4 gateway for the client. On the IPv6 side, our network's border gateway allocates a `/64` IPv6 subnet to the PINOT switch; PINOT selects IPv6 addresses in this subnet that are used by the IPv4 client when communicating with the Internet.

The client's Do53 traffic traverses the PINOT switch and the IPv6 border gateway to reach public DNS resolvers on the Internet. The PINOT switch automatically translates a target DNS resolver address to an IPv6 equivalent and vice versa for the response using pre-installed rules.

From the lists of 11,884 public DNS resolvers [34], we found 374 DNS resolvers that have both IPv4 and IPV6 addresses and thus can answer IPv4 or IPv6 queries correctly. The IPv4/v6 address pairs of these resolvers are stored in the PINOT switch's IP4to6 and IP6to4 tables.

Feasibility. To evaluate if PINOT correctly encrypts, decrypts, and forwards Do53 traffic, We use dig to send DNS queries for A records of ten unique domains randomly selected from the top 1 million domains to each resolver (totalling 3,740 queries), using IPv4 and IPv6 networks. There are 3,387 *consistent* queries, i.e., returning the exact same A records in both settings; the inconsistent responses are caused by either DNS load balancing or resolver-side errors (misconfiguration, etc.). We replay the consistent queries with PINOT running and find all returned A records are consistent with the results that use IPv4 or IPv6 networks.

Latency introduced by IPv6 routing. Though encryption and decryption can be performed at switch hardware rates (e.g., 3.2 Tb/s on our switch), there is one potential source of overhead: the routing paths taken by IPv6 packets and IPv4 packets could be different, affecting latency. We test 1,000 DNS queries and examine the query time. As shown in Figure 5, Though using IPv6 may add up to 30 ms of delay in a query, PINOT does not introduce additional latency in 97% of the cases.

Overall, we conclude that PINOT does not affect the normal use of Do53.

6 RELATED WORK

Cryptographic algorithms in programmable data planes. The ability to support cryptographic algorithms in programmable hardware is important for offloading security and privacy applications to data planes. Previous works mostly focus on using switch CPUs, SmartNICs, or NetFPGAs to implement cryptographic algorithms, but these approaches may have performance, scalability, or compatibility issues [20, 21, 38]. Very few studies have examined using switch ASICs for cryptographic operations. SPINE implements a prototype

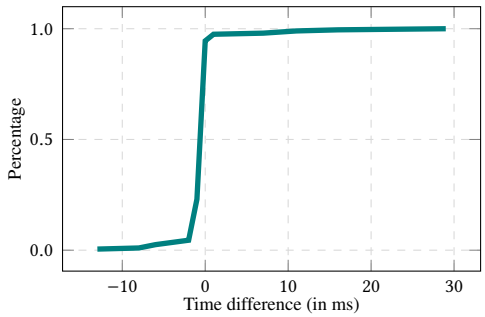

Figure 5: CDF of DNS response time differences.

of SipHash for the BMv2 software model [13]. However, unlike in software, SPINE likely needs at least three passes of the packet on a hardware switch due to resource constraints, degrading the throughput by a factor of three. Similarly, P4-AES [9] requires at least two more passes of a packet on hardware switches. In contrast to SPINE and P4-AES, PINOT is able to fit into switch ASICs and encrypt source IP addresses using a single pass of the packet-processing pipeline on commercial off-the-shelf programmable switches.

Hiding user IP addresses. Network-layer anonymity systems, such as LAP [24], Dovetail [36], HORNET [5], PHI [7], and TARANET [6], typically require multiple ASes along an end-to-end path to cooperate in the protocol and end-users to run specialized software. The involvement of end-users not only further raises deployment barriers, but also introduces human errors [32] that cause privacy failures. None of the systems were implemented on programmable hardware.

Address Hiding Protocol (AHP) [35] and SPINE [13] can conceal users' IP addresses without user participation. In AHP, a trusted network assigns a random IP address to a user from its own IPv4 address space, which poses security issues for small networks. In addition, AHP does not provide packet unlinkability. SPINE encrypts the IP address in every packet using a programmable switch, but requires an additional trusted network at the receiver to decrypt every packet.

DNS privacy. Encrypted DNS such as DNS-over-HTTPS (DoH), DNS-over-TLS (DoT), and DNSCrypt [16, 23, 25] can encrypt DNS queries. However, they put trust in third-party servers that are able to associated client IP addresses to DNS queries, which could become a single point of privacy failure. Anonymized DNSCrypt [14], Oblivious DNS (ODNS) [37], and Oblivious DoH (ODoH) [39] use proxies to hide client IP addresses from third-party resolvers, but requires modifications to DNS clients and infrastructure and introduce relatively high latency. In contrast to other proxy-based solutions, PINOT does not require cooperation from end-users and introduces low proxy performance overhead.

7 CONCLUSION

PINOT is a lightweight in-network anonymity solution that hides users' IP addresses from downstream ASes and destination servers. Utilizing an efficient and secure encryption scheme, PINOT can encrypt IP addresses at hardware switch rates. In contrast to known anonymity solutions, PINOT has a low barrier to deployment, because it requires no cooperation from end-users or any ASes other than the trusted network where it is deployed. We implemented and deployed a prototype of PINOT, and demonstrated PINOT is feasible for improving user privacy in DNS.

REFERENCES

[1] Jawad Ahmed, Hassan Habibi Gharakheili, Qasim Raza, Craig Russell, and Vijay Sivaraman. 2019. Monitoring enterprise DNS queries for detecting data exfiltration from internal hosts. *IEEE Transactions on Network and Service Management* 17, 1 (2019), 265–279.

[2] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, François-Xavier Standaert, John Steinberger, and Elmar Tischhauser. 2012. Key-alternating ciphers in a provable setting: Encryption using a small number of public permutations. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 45–62.

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (July 2014), 87–95.

[4] Caida. 2008. Packet size distribution comparison between Internet links in 1998 and 2008. https://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml. (2008).

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

[6] Chen Chen, Daniele E Asoni, Adrian Perrig, David Barrera, George Danezis, and Carmela Troncoso. 2018. TARANET: Traffic-analysis resistant anonymity at the network layer. In *IEEE European Symposium on Security and Privacy*. IEEE, 137–152.

[7] Chen Chen and Adrian Perrig. 2017. PHI: Path-Hidden Lightweight Anonymity Protocol at Network Layer. In *Privacy Enhancing Technologies*. 100–117.

[8] Shan Chen, Rodolphe Lampe, Jooyoung Lee, Yannick Seurin, and John Steinberger. 2018. Minimizing the two-round Even–Mansour cipher. *Journal of Cryptology* 31, 4 (2018), 1064–1119.

[9] Xiaoqi Chen. 2020. Implementing AES encryption on programmable switches via scrambled lookup tables. In *ACM SIGCOMM Workshop on Secure Programmable Network Infrastructure*. 8–14.

[10] Cisco. 2020. Configuring NAT for IP Address Conservation. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_nat/configuration/xe-16/nat-xe-16-book/iadnat-addr-consv.html. (2020).

[11] Citrix. 2020. Stateless NAT46. https://docs.citrix.com/en-us/netscaler/12/networking/ip-addressing/configuring-network-address-translation/stateless-nat46-translation.html. (2020).

[12] Joan Daemen and Vincent Rijmen. 2006. Two-Round AES Differentials. *IACR Cryptology ePrint Archive* (2006), 39.

[13] Trisha Datta, Nick Feamster, Jennifer Rexford, and Liang Wang. 2019. SPINE: Surveillance Protection in the Network Elements. In *USENIX*

Workshop on Free and Open Communications on the Internet.
[14] Frank Denis. 2020. Anonymized DNSCrypt. https://github.com/DNSCrypt/dnscrypt-proxy/wiki/Anonymized-DNS. (2020).
[15] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. 2016. Key recovery attacks on iterated Even-Mansour encryption schemes. *Journal of Cryptology* 29, 4 (2016), 697–728.
[16] DNSCrypt. 2020. DNSCrypt. https://dnscrypt.info/. (2020).
[17] Jason A Donenfeld. 2017. WireGuard: Next Generation Kernel Network Tunnel. In *Network and Distributed System Security Symposium*.
[18] Orr Dunkelman, Gautham Sekar, and Bart Preneel. 2007. Improved meet-in-the-middle attacks on reduced-round DES. In *International Conference on Cryptology in India*. Springer, 86–100.
[19] Benjamin Greschbach, Tobias Pulls, Laura M. Roberts, Philipp Winter, and Nick Feamster. 2017. The Effect of DNS on Tor's Anonymity. In *Network and Distributed System Security Symposium*.
[20] Frederik Hauser, Marco Häberle, Mark Schmidt, and Michael Menth. 2020. P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN. *IEEE Access* 8 (2020), 139567–139586.
[21] Frederik Hauser, Mark Schmidt, Marco Häberle, and Michael Menth. 2020. P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection with MACsec in P4-Based SDN. *IEEE Access* (2020).
[22] Dominik Herrmann, Christian Banse, and Hannes Federrath. 2013. Behavior-based tracking: Exploiting characteristic patterns in DNS traffic. *Computers & Security* 39 (2013), 17–33.
[23] P. Hoffman and P. McManus. 2018. *DNS Queries over HTTPS (DoH)*. RFC 8484. RFC Editor.
[24] Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Adrian Perrig, Akira Yamada, Samuel C Nelson, Marco Gruteser, and Wei Meng. 2012. LAP: Lightweight anonymity and privacy. In *IEEE Symposium on Security and Privacy*. IEEE, 506–520.
[25] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. 2016. *Specification for DNS over Transport Layer Security (TLS)*. RFC 7858. RFC Editor.
[26] Intel. 2020. Intel Tofino. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html. (2020).
[27] Rodolphe Lampe, Jacques Patarin, and Yannick Seurin. 2012. An asymptotically tight security analysis of the iterated Even-Mansour cipher. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 278–295.
[28] Matthew Luckie, Robert Beverly, Ryan Koga, Ken Keys, Joshua A Kroll, and k claffy. 2019. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *ACM SIGSAC Conference on Computer and Communications Security*. 465–480.
[29] Edgecore Networks. 2020. Edge-core Wedge 100BF-32X. https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335,2019. (2020).
[30] Ivica Nikolić, Lei Wang, and Shuang Wu. 2013. Cryptanalysis of Round-Reduced LED. In *International Workshop on Fast Software Encryption*. Springer, 112–129.
[31] University of Oregon. 2020. Route Views Archive Project. http://archive.routeviews.org/. (2020).
[32] Patrick Howell O'Neill. 2020. The real chink in Tor's armor. https://www.dailydot.com/crime/silk-road-tor-arrests/. (2020).
[33] P4 privacy. 2020. PINOT source code. https://github.com/liangw89/p4privacy/tree/master/pinot. (2020).
[34] Public-dns.info. 2020. Public DNS resolvers. https://public-dns.info/. (2020).
[35] Barath Raghavan, Tadayoshi Kohno, Alex C. Snoeren, and David Wetherall. 2009. Enlisting ISPs to Improve Online Privacy: IP Address Mixing by Default. In *Privacy Enhancing Technologies Symposium*. Springer Berlin Heidelberg, 143–163.
[36] Jody Sankey and Matthew Wright. 2014. Dovetail: Stronger anonymity in next-generation internet routing. In *Privacy Enhancing Technologies Symposium*. Springer, 283–303.
[37] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. 2019. Oblivious DNS: Practical privacy for DNS queries. In *Privacy Enhancing Technologies Symposium*, Vol. 2019. Sciendo, 228–244.
[38] Dominik Scholz, Andreas Oeldemann, Fabien Geyer, Sebastian Gallenmüller, Henning Stubbe, Thomas Wild, Andreas Herkersdorf, and Georg Carle. 2019. Cryptographic Hashing in P4 Data Planes. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 1–6.
[39] Sudheesh Singanamalla, Suphanat Chunhapanya, Marek Vavruša, Tanya Verma, Peter Wu, Marwan Fayed, Kurtis Heimerl, Nick Sullivan, and Christopher Wood. 2020. Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS. (2020). arXiv:cs.CR/2011.10121
[40] Tele2. 2020. Tele2 Speedtest. http://speedtest.tele2.net/. (2020).
[41] Wikipedia. 2020. Substitution–permutation network. https://en.wikipedia.org/wiki/Substitution-permutation_network. (Feb 2020).
[42] Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. 2007. Analysis of privacy disclosure in DNS query. In *International Conference on Multimedia and Ubiquitous Engineering*. IEEE, 952–957.

A APPENDIX

A.1 Security analysis

The security of 2EM depends on the plaintext size n $(32 + l)$. With a larger IPv6 network (i.e., a short network prefix d), PINOT can append more random bits to the original IPv4 source address to produce a longer input. To facilitate discussion, we fix $n = 64$ (i.e., $l = 32$). Note that $2^{\frac{2n}{3}}$ is the lower bound obtained in the information-theoretic model [8, 27]. There is a significant gap between this lower bound and the complexity of the best-known attacks in the computational model. We only consider the best-known attacks for certain types of (computationally bounded) adversaries in this section.

Passive adversary. A passive adversary can only perform the trivial attacks, i.e., an exhaustive key search, which requires brute forcing in the space of 2^{3n} or 2^{192} keys. Such brute-force attacks are clearly infeasible for computationally bounded adversaries.

Known/chosen-plaintext adversary. Non-trivial attacks usually need to consider three important factors: data complexity, memory complexity, and time complexity, where data complexity is the number of ciphertext-plaintext pairs they need to collect, and memory complexity is the number of memory units (n-bit blocks) required during attacks. A known-plaintext adversary may trade memory complexity for data complexity using the man-in-the-middle (MITM) attacks proposed by Andrey *et al.* [2]. A MITM attack only requires knowing a small number of ciphertext-plaintext pairs (e.g., 2); however, its memory complexity is 2^n, i.e., more than 147

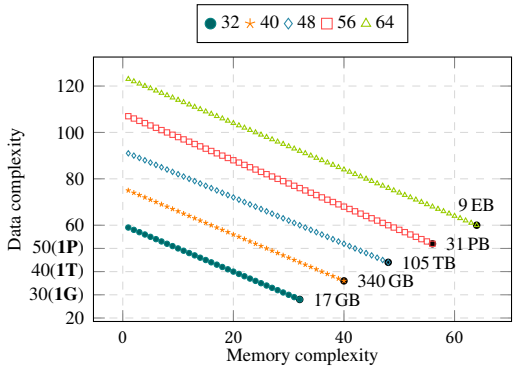

Figure 6: Data complexity vs memory complexity of the key recovery attacks under varied n (32, 40, 48, 56 and 64 bits). The memory complexity increases from 2^1 to 2^n. X-axis and Y-axis are in log2 scale. Data complexity indicates the number of plaintext -ciphertext pairs (i.e., encrypted packets) an adversary needs to collect.

exabytes of memory is required for storing a precomputed table.

The best *known* attacks against 2EM with independent keys and permutations are the key recovery attacks proposed by Dinur *et al.* [15], aiming to lower time complexity. The key recovery attacks also require precomputing a big table, and the memory complexity M and data complexity D approximately satisfy $D = 2^{2(n-1)}/(4M)$. The memory-data trade-offs are shown in Figure 6. Indeed, a powerful adversary may be able and be willing to prepare a large amount of memory. However, *practical attacks are hard.*, as the adversary must obey rate limits. In practice, even a large ISP may not see more than 1 T packets per second (approximately 2.4 Pbps assuming the average IPv4 packet size is 300 bytes). Under this constraint, the adversary still needs to prepare 2^{89} bits of memory for $n = 64$, and more memory if targeting a lower packet rate. The use of random padding also makes attacks harder because the adversary can only choose or know partial plaintexts.

Besides, the adversary cannot perform chosen-plaintext attacks in a majority of networks or ASes because of a lack of capability on source address spoofing (i.e., choosing plaintexts). The Spoofer project examined more than 7 K /24 networks and found about 85% of them implement certain mechanisms (e.g. Source Address Validation) to filter outbound spoofed-source packets [28]. For an active adversary, generating a large volume of traffic from a few hosts, if possible, can easily be flagged as DDoS attacks.

Chosen-ciphertext adversary. Our encryption scheme does not provide malleability so the adversary might be able to manipulate the destination addresses in the return traffic.

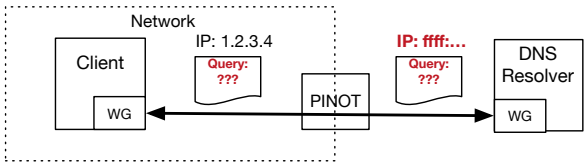

Figure 7: DNS over WireGuard and PINOT.

PINOT may forward a tampered packet based on the decrypted address, or may drop the packet because it does not recognize the decrypted address. In either case the adversary cannot see the decrypted address, and therefore is limited in terms of performing chosen-ciphertext attacks.

2EM alternatives. It is possible to fit the standard ciphers with reduced rounds into the data plane, e.g., 2-round AES and 2-round DES. However, the adversary may break these schemes with low data complexity attacks by exploiting the relation between round keys and algebraic properties of the algorithms [12, 18].

In our setting, a cipher is considered as secure if breaking it requires attacks with high data complexity. Besides, the encrypting and decrypting parties are the same in PINOT, so we do not need to consider key distribution and can store key materials of large sizes. A good alternative to 2EM should use independent round keys to prevent the adversary from exploiting its key schedule. We believe there are other ciphers can be used in lieu of 2EM, and leave exploring 2EM alternatives as future work.

A.2 Obfuscating other IP header fields

We have extended our prototypes to support port encryption by adding a *one-time pad* table that stores random 16-bit one-time pads. PINOT uses the first 16 bit of the generated random padding as a key to fetch the corresponding one-time pad in the one-time pad table, and XOR the one-time pad with the source port in an IPv4 packet. Fetching one-time pads can be done in parallel with IP address encryption, requiring no additional stages. In fact, we can get multiple one-time pads at the same time in one table lookup, and XOR them with different header fields.

A.3 DNS over WireGuard

As discussed in §1.2, since PINOT obfuscates each packet individually, to use PINOT with Do53 or DNSCrypt, the DNS request must fit into one UDP packet. To relax this requirement, we consider sending DNS requests over an emerging VPN tunnel, WireGuard. As shown in Figure 7, The DNS resolver also serves as a VPN peer that only accepts WireGuard VPN connections from registered DNS clients.

WireGuard is a UDP-based, connectionless VPN protocol that will be merged into Linux kernel soon [17]. WireGuard leverages a special mechanism to achieve good IP mobility: it assumes a peer's IP address can change frequently, and uses the source IP address in the *latest* packet received from the peer for future commutations. This enables PINOT to perform per-packet encryption on WireGuard traffic in a stateless way without disrupting connectivity.

Each DNS client is associated with a public key, which could serve as a pseudonym. Though it does see all the DNS requests from a given client based on the public key, the DNS resolver cannot use that information to pinpoint the real client as PINOT obfuscates the client IP address constantly. Without any IP information, The user behavior profiles collected by a malicious resolver are meaningless to other parties (e.g., advertising companies). The use of WireGuard also prevents the trusted network from learning the DNS requests.

We are still working on a full prototype of DNS-over-WireGuard. In this work, we only focus on on the feasibility of per-packet encryption on WireGuard traffic. We set up a WireGuard forwarding server on an AWS EC2 `t2.micro` instance. Our client's WireGuard traffic traverses this server to reach public services on the Internet.

We download 100 randomly selected files with varying sizes (1 KB to 10 GB) from two websites [31, 40]. All files download successfully through WireGuard and PINOT, and the SHA1 hash of every file matches that of equivalent downloads directly using the IPv4 network. Overall, we conclude that per-packet encryption does not affect the normal use of WireGuard. We leave performance comparison between DNS-over-WireGaurd, DoH, DoT, and other DNS privacy-enhancing mechanisms as future work.