Embark: Securely Outsourcing Middleboxes to the Cloud

Chang Lan, Justine Sherry, Raluca Ada Popa, and Sylvia Ratnasamy, ***University of California, Berkeley;*** **Zhi Liu,** ***Tsinghua University***

https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/lan

This paper is included in the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16).

March 16–18, 2016 • Santa Clara, CA, USA

ISBN 978-1-931971-29-4

Open access to the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16) is sponsored by USENIX.

Embark: Securely Outsourcing Middleboxes to the Cloud

Chang Lan Justine Sherry Raluca Ada Popa Sylvia Ratnasamy Zhi Liu*
UC Berkeley **Tsinghua University*

Abstract

It is increasingly common for enterprises and other organizations to outsource network processing to the cloud. For example, enterprises may outsource firewalling, caching, and deep packet inspection, just as they outsource compute and storage. However, this poses a threat to enterprise confidentiality because the cloud provider gains access to the organization's traffic.

We design and build Embark, the first system that enables a cloud provider to support middlebox outsourcing while maintaining the client's confidentiality. Embark encrypts the traffic that reaches the cloud and enables the cloud to process the encrypted traffic without decrypting it. Embark supports a wide-range of middleboxes such as firewalls, NATs, web proxies, load balancers, and data exfiltration systems. Our evaluation shows that Embark supports these applications with competitive performance.

1 Introduction

Middleboxes such as firewalls, NATs, and proxies, have grown to be a vital part of modern networks, but are also widely recognized as bringing significant problems including high cost, inflexibility, and complex management. These problems have led both research and industry to explore an alternate approach: moving middlebox functionality out of dedicated boxes and into software applications that run multiplexed on commodity server hardware [53, 52, 54, 29, 37, 28, 27, 14, 8]. This approach – termed Network Function Virtualization (NFV) in industry – promises many advantages including the cost benefits of commodity infrastructure and outsourced management, the efficiency of statistical multiplexing, and the flexibility of software solutions. In a short time, NFV has gained a significant momentum with over 270 industry participants [27] and a number of emerging product offerings [1, 7, 6].

Leveraging the above trend, several efforts are exploring a new model for middlebox deployment in which a third-party offers middlebox processing as a *service*. Such a service may be hosted in a public cloud [54, 13, 17] or in private clouds embedded within an ISP infrastructure [14, 11]. This service model allows customers such as enterprises to "outsource" middleboxes from their networks entirely, and hence promises many of the known benefits of cloud computing such as decreased costs and ease of management.

However, outsourcing middleboxes brings a new challenge: the confidentiality of the traffic. Today, in order to process an organization's traffic, the cloud sees the traffic *unencrypted*. This means that the cloud now has access to potentially sensitive packet payloads and headers. This is worrisome considering the number of documented data breaches by cloud employees or hackers [23, 60]. Hence, an important question is: can we enable a third party to process traffic for an enterprise, *without seeing the enterprise's traffic*?

To address this question, we designed and implemented Embark[1], the first system to allow an enterprise to outsource a wide range of enterprise middleboxes to a cloud provider, while keeping its network traffic confidential. Middleboxes in Embark operate directly over *encrypted* traffic without decrypting it.

In previous work, we designed a system called BlindBox to operate on encrypted traffic for a *specific* class of middleboxes: Deep Packet Inspection (DPI) [55] – middleboxes that examine only the payload of packets. However, BlindBox is far from sufficient for this setting because (1) it has a restricted functionality that supports too few of the middleboxes typically outsourced, and (2) it has prohibitive performance overheads in some cases. We elaborate on these points in §2.4.

Embark supports a wide range of middleboxes with practical performance. Table 1 shows the relevant middleboxes and the functionality Embark provides. Embark achieves this functionality through a combination of systems and cryptographic innovations, as follows.

From a cryptographic perspective, Embark provides a new and fast encryption scheme called PrefixMatch to enable the provider to perform prefix matching (*e.g.*, if an IP address is in the subdomain 56.24.67.0/16) or port range detection (*e.g.*, if a port is in the range 1000-2000). PrefixMatch allows matching an encrypted packet field against an encrypted prefix or range using the same operators as for unencrypted data: $\geq$ and prefix equality. At the same time, the comparison operators do not work when used between encrypted packet fields. Prior to PrefixMatch, there was no mechanism that provided the functionality, performance, and security needed in our setting. The closest practical encryption schemes are Order-Preserving Encryption (OPE) [21, 48]. However, we show that these schemes are four orders of magnitude slower than

[1] This name comes from "mb" plus "ark", a shortcut for middlebox and a synonym for protection, respectively.

	Middlebox	Functionality	Support	Scheme
L3/L4 Header	IP Firewall [66]	$(SIP, DIP, SP, DP, P) \in (SIP[], DIP[], SP[], DP[], P)$ $\Leftrightarrow \mathsf{Enc}(SIP, DIP, SP, DP, P) \in \mathsf{Enc}(SIP[], DIP[], SP[], DP[], P)$	Yes	PrefixMatch
	NAT (NAPT) [57]	$(SIP_1,DIP_1,SP_1,DP_1,P_1) = (SIP_2,DIP_2,SP_2,DP_2,P_2)$ $\Rightarrow \mathsf{Enc}(SIP_1,SP_1) = \mathsf{Enc}(SIP_2,SP_2)$ $\mathsf{Enc}(SIP_1,SP_1) = \mathsf{Enc}(SIP_2,SP_2) \Rightarrow (SIP_1,SP_1) = (SIP_2,SP_2)$	Yes	PrefixMatch
	L3 LB (ECMP) [58]	$(SIP_1,DIP_1,SP_1,DP_1,P_1) = (SIP_2,DIP_2,SP_2,DP_2,P_2)$ $\Leftrightarrow \mathsf{Enc}(SIP_1,DIP_1,SP_1,DP_1,P_1) = \mathsf{Enc}(SIP_2,DIP_2,SP_2,DP_2,P_2)$	Yes	PrefixMatch
	L4 LB [4]	$(SIP_1,DIP_1,SP_1,DP_1,P_1) = (SIP_2,DIP_2,SP_2,DP_2,P_2)$ $\Leftrightarrow \mathsf{Enc}(SIP_1,DIP_1,SP_1,DP_1,P_1) = \mathsf{Enc}(SIP_2,DIP_2,SP_2,DP_2,P_2)$	Yes	PrefixMatch
HTTP	HTTP Proxy / Cache [25, 4, 10]	$\mathsf{Match}(\text{Request-URI, HTTP Header})$ $= \mathsf{Match}'(\mathsf{Enc}(\text{Request-URI}), \mathsf{Enc}(\text{HTTP Header}))$	Yes	KeywordMatch
Deep Packet Inspection (DPI)	Parental Filter [10]	$\mathsf{Match}(\text{Request-URI, HTTP Header})$ $= \mathsf{Match}'(\mathsf{Enc}(\text{Request-URI}), \mathsf{Enc}(\text{HTTP Header}))$	Yes	KeywordMatch
	Data Exfiltration / Watermark Detection [56]	$\mathsf{Match}(\text{Watermark, Stream})$ $= \mathsf{Match}'(\mathsf{Enc}(\text{Watermark}), \mathsf{Enc}(\text{Stream}))$	Yes	KeywordMatch
	Intrusion Detection [59, 47]	$\mathsf{Match}(\text{Keyword, Stream}) =$ $\mathsf{Match}'(\mathsf{Enc}(\text{Keyword}), \mathsf{Enc}(\text{Stream}))$	Yes	KeywordMatch
		$\mathsf{RegExpMatch}(\text{RegExp, Stream})$ $= \mathsf{RegExpMatch}'(\mathsf{Enc}(\text{RegExp}), \mathsf{Enc}(\text{Stream}))$	Partially	KeywordMatch
		Run scripts, cross-flow analysis, or other advanced (e.g. statistical) tools	No	-

Table 1: Middleboxes supported by Embark. The second column indicates an encryption functionality that is sufficient to support the core functionality of the middlebox. Appendix §A demonstrates this sufficiency. "Support" indicates whether Embark supports this functionality and "Scheme" is the encryption scheme Embark uses to support it. **Legend:** Enc denotes a generic encryption protocol, *SIP* = source IP address, *DIP* = destination IP, *SP* = source port, *DP* = destination port, *P* = protocol, $E[]$ = a range of E, $\Leftrightarrow$ denotes "if and only if", $\mathsf{Match}(x,s)$ indicates if x is a substring of s, and Match' is the encrypted equivalent of Match. Thus, (SIP, DIP, SP, DP, P) denotes the tuple describing a connection.

PrefixMatch making them infeasible for our network setting. At the same time, PrefixMatch provides stronger security guarantees than these schemes: PrefixMatch does not reveal the order of encrypted packet fields, while OPE reveals the total ordering among all fields. We designed PrefixMatch specifically for Embark's networking setting, which enabled such improvements over OPE.

From a systems design perspective, one of the key insights behind Embark is to keep packet formats and header classification algorithms unchanged. An encrypted IP packet is structured just as a normal IP packet, with each field (e.g., source address) containing an encrypted value of that field. This strategy ensures that encrypted packets never appear invalid, e.g., to existing network interfaces, forwarding algorithms, and error checking. Moreover, due to PrefixMatch's functionality, header-based middleboxes can run existing highly-efficient packet classification algorithms [34] without modification, which are among the more expensive tasks in software middleboxes [52]. Furthermore, even software-based NFV deployments use some hardware forwarding components, *e.g.* NIC multiqueue flow hashing [5], 'whitebox' switches [12], and error detection in NICs and switches [5, 2]; Embark is also compatible with these.

Embark's unifying strategy was to reduce the core functionality of the relevant middleboxes to two basic opera-

tions over different fields of a packet: prefix and keyword matching, as listed in Table 1. This results in an encrypted packet that *simultaneously* supports these middleboxes.

We implemented and evaluated Embark on EC2. Embark supports the core functionality of a wide-range of middleboxes as listed in Table 1, and elaborated in Appendix A. In our evaluation, we showed that Embark supports a real example for each middlebox category in Table 1. Further, Embark imposes negligible throughput overheads at the service provider: for example, a single-core firewall operating over encrypted data achieves 9.8Gbps, equal to the same firewall over unencrypted data. Our enterprise gateway can tunnel traffic at 9.6 Gbps on a single core; a single server can easily support 10Gbps for a small-medium enterprise.

2 Overview

In this section, we present an overview of Embark.

2.1 System Architecture

Embark uses the same architecture as APLOMB [54], a system which redirects an enterprise's traffic to the cloud for middlebox processing. Embark augments this architecture with confidentiality protection.

In the APLOMB setup, there are two parties: the enterprise(s) and the service provider or cloud (SP). The enterprise runs a gateway (GW) which sends traffic to middleboxes (MB) running in the cloud; in practice, this cloud may be either a public cloud service (such as EC2), or an ISP-supported service running at a Central Office (CO).

We illustrate the two redirection setups from APLOMB in Fig. 1. The first setup, in Fig. 1(a), occurs when the enterprise communicates with an external site: traffic goes to the cloud and back before it is sent out to the Internet. It is worth mentioning that APLOMB allows an optimization that saves on bandwidth and latency relative to Fig. 1(a): the traffic from SP can go directly to the external site and does not have to go back through the gateway. Embark does not allow this optimization fundamentally: the traffic from SP is encrypted and cannot be understood by an external site. Nonetheless, as we demonstrate in §6, for ISP-based deployments this overhead is negligible. For traffic within the same enterprise, where the key is known by two gateways owned by the same company, we can support the optimization as shown in Fig. 1(b).

We do not delve further into the details and motivation of APLOMB's setup, but instead refer the reader to [54].

2.2 Threat Model

Clients adopt cloud services for decreased cost and ease of management. Providers are known and trusted to provide good service. However, while clients trust cloud providers to perform their services correctly, there is an increasing concern that cloud providers may access or leak confidential data in the process of providing service. Reports in the popular press describe companies selling customer data to marketers [20], disgruntled employees snooping or exporting data [16], and hackers gaining access to data on clouds [60, 23]. This type of threat is referred to as an 'honest but curious' or 'passive' [33] attacker: a party who is trusted to handle the data and deliver service correctly, but who looks at the data, and steals or exports it. Embark aims to stop these attackers. Such an attacker differs from the 'active' attacker, who manipulates data or deviates from the protocol it is supposed to run [33]. We consider that such a passive attacker has gained access to *all the data at SP*. This includes any traffic and communication SP receives from the gateway, any logged information, cloud state, and so on.

We assume that the gateways are managed by the enterprise and hence trusted; they do not leak information.

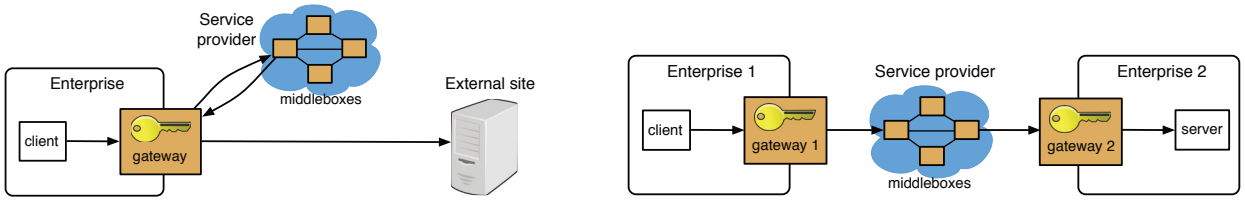
Some middleboxes (such as intrusion or exfiltration detection) have a threat model of their own about the two endpoints communicating. For example, intrusion detection assumes that one of the endpoints could misbehave, but at most one of them misbehaves [47]. We preserve these threat models unchanged. These applications rely on the middlebox to detect attacks in these threat models. Since we assume the middlebox executes its functions correctly and Embark preserves the functionality of these middleboxes, these threat models are irrelevant to the protocols in Embark, and we will not discuss them again.

2.3 Encryption Overview

To protect privacy, Embark *encrypts the traffic* passing through the service provider (SP). Embark encrypts both the header and the payload of each packet, so that SP does not see this information. We encrypt headers because they contain information about the endpoints.

Embark also provides the cloud provider with a set of *encrypted rules*. Typically, header policies like firewall rules are generated by a local network administrator. Hence, the gateway knows these rules, and these rules may or may not be hidden from the cloud. DPI and filtering policies, on the other hand, may be private to the enterprise (as in exfiltration policies), known by both parties (as in public blacklists), or known only by the cloud provider (as in proprietary malware signatures). We discuss how rules are encrypted, generated and distributed given these different trust settings in §4.2.

As in Fig. 1, the gateway has a secret key k; in the setup with two gateways, they share the same secret key. At setup time, the gateway generates the set of encrypted rules using k and provides them to SP. Afterwards, the gateway encrypts all traffic going to the service provider using Embark's encryption schemes. The middleboxes at SP process encrypted traffic, comparing the traffic against the encrypted rules. After the processing, the middleboxes will produce encrypted traffic which SP sends back to the



(a) Enterprise to external site communication

(b) Enterprise to enterprise communication

Figure 1: System architecture. APLOMB and NFV system setup with Embark encryption at the gateway. The arrows indicate traffic from the client to the server; the response traffic follows the reverse direction.

gateway. The gateway decrypts the traffic using the key k.

Throughout this process, middleboxes at SP handle only encrypted traffic and never access the decryption key. On top of Embark's encryption, the gateway can use a secure tunneling protocol, such as SSL or IPSec to secure the communication to SP.

Packet encryption. A key idea is to encrypt packets *field-by-field*. For example, an encrypted packet will contain a source address that is an encryption of the original packet's source address. We ensure that the encryption has the same size as the original data, and place any additional encrypted information or metadata in the options field of a packet. Embark uses three encryption schemes to protect the privacy of each field while allowing comparison against encrypted rules at the cloud:

- Traditional AES: provides strong security and no computational capabilities.
- KeywordMatch: allows the provider to detect if an encrypted value in the packet is equal to an encrypted rule; does not allow two encrypted values to be compared to each other.
- PrefixMatch: allows the provider to detect whether or not an encrypted value lies in a range of rule values – e.g. addresses in 128.0.0.0/24 or ports between 80-96.

We discuss these cryptographic algorithms in §3.

For example, we encrypt IP addresses using PrefixMatch. This allows, e.g., a firewall to check whether the packet's source IP belongs to a prefix known to be controlled by a botnet – but without learning what the actual source IP address is. We choose which encryption scheme is appropriate for each field based on a classification of middlebox capabilities as in Table 1. In the same table, we classify middleboxes as operating only over L3/L4 headers, operating only over L3/L4 headers and HTTP headers, or operating over the entire packet including arbitrary fields in the connection bytestream (DPI). We revisit each category in detail in §5.

All encrypted packets are IPv6 because PrefixMatch requires more than 32 bits to encode an encrypted IP address and because we expect more and more service providers to be moving to IPv6 by default in the future. This is a trivial requirement because it is easy to convert from IPv4 to IPv6 (and back) [42] at the gateway. Clients may continue using IPv4 and the tunnel connecting the gateway to the provider may be either v4 or v6.

Example. Fig. 2 shows the end-to-end flow of a packet through three example middleboxes in the cloud, each middlebox operating over an encrypted field. Suppose the initial packet was IPv4. First, the gateway converts the packet from IPv4 to IPv6 and encrypts it. The options field now contains some auxiliary information which will help the gateway decrypt the packet later. The packet passes through the firewall which tries to match the encrypted information from the header against its encrypted rule, and decides to allow the packet. Next, the exfiltration device checks for any suspicious (encrypted) strings in data encrypted for DPI and not finding any, it allows the packet to continue to the NAT. The NAT maps the source IP address to a different IP address. Back at the enterprise, the gateway decrypts the packet, except for the source IP written by the NAT. It converts the packet back to IPv4.

2.4 Architectural Implications and Comparison to BlindBox

When compared to BlindBox, Embark provides broader functionality and better performance. Regarding functionality, BlindBox [55] enables equality-based operations on encrypted payloads of packets, which supports certain DPI devices. However, this excludes middleboxes such as firewalls, proxies, load balancers, NAT, and those DPI devices that also examine packet headers, because these need an encryption that is compatible with packet headers and/or need to perform range queries or prefix matching.

The performance improvement comes from the different architectural setting of Embark, which provides a set of interesting opportunities. In BlindBox, two arbitrary user endpoints communicate over a modified version of HTTPS. BlindBox requires 97 seconds to perform the initial handshake, which must be performed for every new connection. However, in the Embark context, this exchange can be performed just once at the gateway because the connection between the gateway and the cloud provider is long-lived. Consequently, there is no per-user-connection overhead.

The second benefit is increased deployability. In Embark, the gateway encrypts traffic whereas in BlindBox

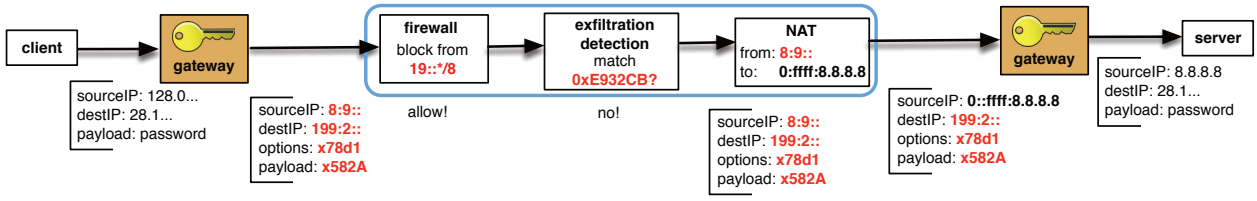


Figure 2: Example of packet flow through a few middleboxes. Red in bold indicates encrypted data.

the end hosts do. Hence, deployability improves because the end hosts do not need to be modified.

Finally, security improves in the following way. BlindBox has two security models: a stronger one to detect rules that are 'exact match' substrings, and a weaker one to detect rules that are regular expressions. The more rules there are, the higher the per-connection setup cost is. Since there is no per-connection overhead in Embark, we can afford having more rules. Hence, we convert many regular expressions to a set of exact-match strings. For example /hello[1-3]/ is equivalent to exact matches on "hello1", "hello2", "hello3". Nonetheless, many regular expressions remain too complex to do so – if the set of potential exact matches is too large, we leave it as a regular expression. As we show in §6, this approach halves the number of rules that require using the weaker security model, enabling more rules in the stronger security model.

In the rest of the paper, we do not revisit these architectural benefits, but focus on Embark's new capabilities that allow us to outsource a *complete* set of middleboxes.

2.5 Security guarantees

We formalize and prove the overall guarantees of Embark in our extended paper. In this version, we provide only a high-level description. Embark hides the values of header and payload data, but reveals some information desired for middlebox processing. The information revealed is the union of the information revealed by PrefixMatch and KeywordMatch, as detailed in §3. Embark reveals more than is strictly necessary for the functionality, but it comes close to this necessary functionality. For example, a firewall learns if an encrypted IP address matches an encrypted prefix, without learning the value of the IP address or the prefix. A DPI middlebox learns whether a certain byte offset matches any string in a DPI ruleset.

3 Cryptographic Building Blocks

In this section, we present the building blocks Embark relies on. Symmetric-key encryption (based on AES) is well known, and we do not discuss it here. Instead, we briefly discuss KeywordMatch (introduced by [55], to which we refer the reader for details) and more extensively discuss PrefixMatch, a new cryptographic scheme we designed for this setting. When describing these schemes, we refer to the encryptor as the gateway whose secret key is k and to the entity computing on the encrypted data as the service provider (SP).

3.1 KeywordMatch

KeywordMatch is an encryption scheme using which SP can check if an encrypted rule (the "keyword") matches by equality an encrypted string. For example, given an encryption of the rule "malicious", and a list of encrypted strings [Enc("alice"), Enc("malicious"), Enc("alice")], SP can detect that the rule matches the second string, but it does not learn anything about the first and third strings, not even that they are equal to each other. KeywordMatch provides typical searchable security guarantees, which are well studied: at a high level, given a list of encrypted strings, and an encrypted keyword, SP does not learn anything about the encrypted strings, other than which strings match the keyword. The encryption of the strings is *randomized*, so it does not leak whether two encrypted strings are equal to each other, unless, of course, they both match the encrypted keyword. We use the scheme from [55] and hence do not elaborate on it.

3.2 PrefixMatch

Many middleboxes perform detection over *prefixes* or *ranges* of IP addresses or port numbers (i.e. packet classification). To illustrate PrefixMatch, we use IP addresses (IPv6), but the scheme works with ports and other value domains too. For example, a network administrator might wish to block access to all servers hosted by MIT, in which case the administrator would block access to the prefix 0::ffff:18.0.0.0/104, *i.e.*, 0::ffff:18.0.0.0/104–0::ffff:18.255.255.255/104. PrefixMatch enables a middlebox to tell whether an encrypted IP address v lies in an encrypted range $[s_1, e_1]$, where s_1 = 0::ffff:18.0.0.0/104 and e_1 = 0::ffff:18.255.255.255/104. At the same time, the middlebox does not learn the values of v, s_1, or e_1.

One might ask whether PrefixMatch is necessary, or one can instead employ KeywordMatch using the same expansion technique we used for some (but not all) regexps in §2.4. To detect whether an IP address is in a range, one could enumerate all IP addresses in that range and perform an equality check. However, the overhead of using this technique for common network ranges such as firewall rules is prohibitive. For our own department network, doing so would convert our IPv6 and IPv4 firewall rule set of only 97 range-based rules to

2^{238} exact-match rules; looking only at IPv4 rules would still lead to 38M exact-match rules. Hence, for efficiency, we need a new scheme for matching ranges.

Requirements. Supporting the middleboxes from Table 1 and meeting our system security and performance requirements entail the following requirements in designing PrefixMatch. First, PrefixMatch must allow for direct order comparison (i.e., using $\leq/\geq$) between an encrypted value $\mathsf{Enc}(v)$ and the encrypted endpoints $\overline{s_1}$ and $\overline{e_1}$ of a range, $[s_1, e_1]$. This allows existing packet classification algorithms, such as tries, area-based quadtrees, FIS-trees, or hardware-based algorithms [34], to run unchanged.

Second, to support the functionality of NAT as in Table 1, $\mathsf{Enc}(v)$ must be *deterministic within a flow*. Recall that a flow is a 5-tuple of source IP and port, destination IP and port, and protocol. Moreover, the encryption corresponding to two pairs (IP_1, port_1) and (IP_2, port_2) must be injective: if the pairs are different, their encryption should be different.

Third, for security, we require that nothing leaks about the value v other than what is needed by the functionality above. Note that Embark's middleboxes do not need to know the order between two encrypted values $\mathsf{Enc}(v_1)$ and $\mathsf{Enc}(v_2)$, but only comparison to endpoints; hence, PrefixMatch does not leak such order information. PrefixMatch also provides protection for the endpoints of ranges: SP should not learn their values, and SP should not learn the ordering of the intervals. Further, note that the NAT does not require that $\mathsf{Enc}(v)$ be deterministic across flows; hence, PrefixMatch hides whether two IP addresses encrypted as part of different flows are equal or not. In other words, PrefixMatch is randomized across flows.

Finally, both encryption (performed at the gateway) and detection (performed at the middlebox) should be practical for typical middlebox line rates. Our PrefixMatch encrypts in $< 0.5\mu$s per value (as we discuss in §6), and the detection is the same as regular middleboxes based on the $\leq/\geq$ operators.

Functionality. PrefixMatch encrypts a set of ranges or prefixes $P_1, \ldots, P_n$ into a set of encrypted prefixes. The encryption of a prefix P_i consists of one or more encrypted prefixes: $\overline{P_{i,1}}\ldots,\overline{P_{i,n_i}}$. Additionally, PrefixMatch encrypts a value v into an encrypted value $\mathsf{Enc}(v)$. These encryptions have the property that, for all i,

$$v \in P_i \Leftrightarrow \mathsf{Enc}(v) \in \overline{P_{i,1}} \cup \cdots \cup \overline{P_{i,n_i}}.$$

In other words, the encryption preserves prefix matching.

For example, suppose that encrypting P = 0::ffff:18.0.0.0/104 results in one encrypted prefix $\overline{P}$ = 1234::/16, encrypting v_1 = 0::ffff:18.0.0.2 results in $\overline{v_1}$ = 1234:db80:85a3:0:0:8a2e:37a0:7334, and encrypting v_2 = 0::ffff:19.0.0.1 results in $\overline{v_2}$ = dc2a:108f:1e16:992e:a53b:43a3:00bb:d2c2. We can see that $\overline{v_1} \in \overline{P}$ and $\overline{v_2} \notin \overline{P}$.

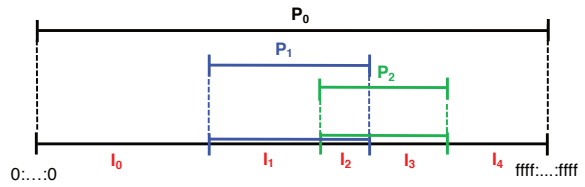


Figure 3: Example of prefix encryption with PrefixMatch.

3.2.1 Scheme

PrefixMatch consists of two algorithms: EncryptPrefixes to encrypt prefixes/ranges and EncryptValue to encrypt a value v.

Prefixes' Encryption. PrefixMatch takes as input a set of prefixes or ranges $P_1 = [s_1, e_1], \ldots, P_n = [s_n, e_n]$, whose endpoints have size len bits. PrefixMatch encrypts each prefix into a set of encrypted prefixes: these prefixes are prefix_len bits long. As we discuss below, the choice of prefix_len depends on the maximum number of prefixes to be encrypted. For example, prefix_len = 16 suffices for a typical firewall rule set.

Consider all the endpoints s_i and e_i laid out on an axis in increasing order as in Fig. 3. Add on this axis the endpoints of P_0, the smallest and largest possible values, 0 and $2^{\text{len}} - 1$. Consider all the non-overlapping intervals formed by each consecutive pair of such endpoints. Each interval has the property that all points in that interval belong to the same set of prefixes. For example, in Fig. 3, there are two prefixes to encrypt: P_1 and P_2. PrefixMatch computes the intervals $I_0, \ldots, I_4$. Two or more prefixes/ranges that overlap in exactly one endpoint define a one-element interval. For example, consider encrypting these two ranges [13::/16, 25::/16] and [25::/16, 27::/16]; they define three intervals: [13::/16, 25::/16-1], [25::/16, 25::/16], [25::/16+1, 27::/16].

Each interval belongs to a set of prefixes. Let $\mathsf{prefixes}(I)$ denote the prefixes of interval I. For example, $\mathsf{prefixes}(I_2) = \{P_0, P_1, P_2\}$.

PrefixMatch now assigns an encrypted prefix to each interval. The encrypted prefix is simply a *random* number of size prefix_len. Each interval gets a different random value, except for intervals that belong to the same prefixes. For example, in Fig. 3, intervals I_0 and I_4 receive the same random number because $\mathsf{prefixes}(I_0) = \mathsf{prefixes}(I_4)$.

When a prefix overlaps partially with another prefix, it will have more than one encrypted prefix because it is broken into intervals. For example, I_1 was assigned a random number of 0x123c and I_2 of 0xabcc. The encryption of P_1 in Fig. 3 will be the pair ($123c::/16$, $abcc::/16$).

Since the encryption is a random prefix, the encryption does not reveal the original prefix. Moreover, the fact that intervals pertaining to the same set of prefixes receive the same encrypted number hides where an encrypted value matches, as we discuss below. For example, for an IP address v that does not match either P_1 or P_2, the cloud

provider will not learn whether it matches to the left or to the right of $P_1 \cup P_2$ because I_0 and I_4 receive the same encryption. The only information it learns about v is that v does not match either P_1 or P_2.

We now present the EncryptPrefixes procedure, which works the same for prefixes or ranges.

EncryptPrefixes (P_1, ..., P_n, prefix_len, len):

1: Let s_i and e_i be the endpoints of P_i. // $P_i = [s_i, e_i]$
2: Assign $P_0 \leftarrow [0, 2^{\text{len}} - 1]$
3: Sort all endpoints in $\cup_i P_i$ in increasing order
4: Construct non-overlapping intervals I_0, ... , I_m from the endpoints as explained above. For each interval I_i, compute prefixes(I_i), the list of prefixes $P_{i_1}, ..., P_{i_m}$ that contain I_i.
5: Let $\overline{I_0}, ..., \overline{I_m}$ each be a distinct random value of size prefix_len.
6: For all i, j with $i < j$ if prefixes(I_i) = prefixes(I_j), set $\overline{I_j} \leftarrow \overline{I_i}$
7: The encryption of P_i is $\overline{P_i} = \{\overline{I_j}/\text{prefix\_len}, \text{ for all } j \text{ s.t. } P_i \in \text{prefixes}(I_j)\}$. The encrypted prefixes are output sorted by value (as a means of randomization).
8: Output $\overline{P_1}, ..., \overline{P_n}$ and the *interval map* $[I_i \rightarrow \overline{I_i}]$

Value Encryption. To encrypt a value v, PrefixMatch locates the one interval I such that $v \in I$. It then looks up $\overline{I}$ in the interval map computed by EncryptPrefixes and sets $\overline{I}$ to be the prefix of the encryption of v. This ensures that the encrypted v, $\overline{v}$, matches $\overline{I}/\text{prefix\_len}$. The suffix of v is chosen at random. The only requirement is that it is deterministic. Hence, the suffix is chosen based on a pseudorandom function [32], $\text{prf}^{\text{suffix\_len}}$, seeded in a given seed seed, where suffix_len = len − prefix_len. As we discuss below, the seed used by the gateway depends on the 5-tuple of a connection (SIP, SP, DIP, DP, P).

For example, if v is 0::ffff:127.0.0.1, and the assigned prefix for the matched interval is $abcd :: /16$, a possible encryption given the ranges encrypted above is $\text{Enc}(v) = abcd : ef01 : 2345 : 6789 : abcd : ef01 : 2345 : 6789$. Note that the encryption does not retain any information about v other than the interval it matches in because the suffix is chosen (pseudo)randomly. In particular, given two values v_1 and v_2 that match the same interval, the order of their encryptions is arbitrary. Thus, PrefixMatch does not reveal order.

EncryptValue (seed, v, suffix_len, interval map):

1: Run binary search on interval map to locate the interval I such that $v \in I$.
2: Lookup $\overline{I}$ in the interval map.
3: Output

$$\text{Enc}(v) = \overline{I} \| \text{prf}^{\text{suffix\_len}}_{\text{seed}}(v) \quad (1)$$

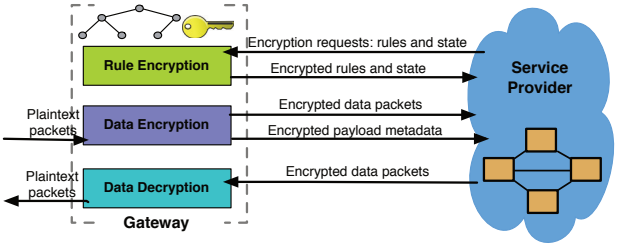


Figure 4: Communication between the cloud and gateway services: rule encryption, data encryption, and data decryption.

Comparing encrypted values against rules. Determining if an encrypted value matches an encrypted prefix is straightforward: the encryption preserves the prefix and a middlebox can use the regular $\leq/\geq$ operators. Hence, a regular packet classification can be run at the firewall with no modification. Comparing different encrypted values that match the same prefix is meaningless, and returns a random value.

3.2.2 Security Guarantees

PrefixMatch hides the prefixes and values encrypted with EncryptPrefixes and EncryptValue. PrefixMatch reveals matching information desired to enable functionality at the cloud provider. Concretely, the cloud provider learns the number of intervals and which prefixes overlap in each interval, but no additional information on the size, order or endpoints of these intervals. Moreover, for every encrypted value v, it learns the indexes of the prefixes that contain v (which is the functionality desired of the scheme), but no other information about v. For any two encrypted values $\text{Enc}(v)$ and $\text{Enc}(v')$, the cloud provider learns if they are equal only if they are encrypted as part of the same flow (which is the functionality desired for the NAT), but it does not learn any other information about their value or order. Hence, PrefixMatch leaks less information than order-preserving encryption, which reveals the order of encrypted prefixes/ranges.

Since EncryptValue is seeded in a per-connection identifier, an attacker cannot correlate values across flows. Essentially, there is a different key per flow. In particular, even though EncryptValue is deterministic within a flow, it is randomized across flows: for example, the encryption of the same IP address in different flows is different because the seed differs per flow.

We formalize and prove the security guarantees of PrefixMatch in our extended paper.

4 Enterprise Gateway

The gateway serves two purposes. First, it redirects traffic to/from the cloud for middlebox processing. Second, it provides the cloud with encryptions of rulesets. Every gateway is configured statically to tunnel traffic to a fixed IP address at a single service provider point of presence. A gateway can be logically thought of as three services: the

rule encryption service, the pipeline from the enterprise to the cloud (Data encryption), and the pipeline from the cloud to the enterprise (Data decryption). All three services share access to the PrefixMatch interval map and the private key k. Fig. 4 illustrates these three services and the data they send to and from the cloud provider.

We design the gateway with two goals in mind:

Format-compatibility: in converting plaintext traffic to encrypted traffic, the encrypted data should be structured in such a way that the traffic *appears as normal IPv6 traffic* to middleboxes performing the processing. Format-compatibility allows us to leave fast-path operations unmodified not only in middlebox software, but also in hardware components like NICs and switches; this results in good performance at the cloud.

Scalability and Low Complexity: the gateway should perform only inexpensive per-packet operations and should be parallelizable. The gateway should require only a small amount of configuration.

4.1 Data Encryption and Decryption

As shown in Table 1, we categorize middleboxes as Header middleboxes, which operate only on IP and transport headers; DPI middleboxes, which operate on arbitrary fields in a connection bytestream; and HTTP middleboxes, which operate on values in HTTP headers (these are a subclass of DPI middleboxes). We discuss how each category of data is encrypted/decrypted in order to meet middlebox requirements as follows.

4.1.1 IP and Transport Headers

IP and Transport Headers are encrypted field by field (*e.g.*, a source address in an input packet results in an encrypted source address field in the output packet) with PrefixMatch. We use PrefixMatch for these fields because many middleboxes perform analysis over prefixes and ranges of values – e.g., a firewall may block all connections from a restricted IP prefix.

To encrypt a value with PrefixMatch's EncryptValue, the gateway seeds the encryption with $\mathsf{seed} = \mathsf{prf}_k(SIP, SP, DIP, DP, P)$, a function of both the key and connection information using the notation in Table 1. Note that in the system setup with two gateways, the gateways generate the same encryption because they share k.

When encrypting IP addresses, two different IP addresses must not map to the same encryption because this breaks the NAT. To avoid this problem, encrypted IP addresses in Embark must be IPv6 because the probability that two IP addresses get assigned to the same encryption is negligibly low. The reason is that each encrypted prefix contains a large number of possible IP addresses. Suppose we have n distinct firewall rules, m flows and a len-bit space, the probability of a collision is approximately:

$$1 - e^{\frac{-m^2(2n+1)}{2^{\mathsf{len}+1}}} \quad (2)$$

Therefore, if $\mathsf{len} = 128$ (which is the case when we use IPv6), the probability is negligible in a realistic setting.

When encrypting ports, it is possible to get collisions since the port field is only 16-bit. However, this will not break the NAT's functionality as long as the IP address does not collide, because NATs (and other middleboxes that require injectivity) consider both IP addresses and ports. For example, if we have two flows with source IP and source ports of (SIP, SP_1) and (SIP, SP_2) with $SP_1 \neq SP_2$, the encryption of SIP will be different in the two flows because the encryption is seeded in the 5-tuple of a connection. As we discuss in Appendix A, the NAT table can be larger for Embark, but the factor is small in practice.

Decryption. PrefixMatch is not reversible. To enable packet decryption, we store the AES-encrypted values for the header fields in the IPv6 options header. When the gateway receives a packet to decrypt, if the values haven't been rewritten by the middlebox (*e.g.*, NAT), it decrypts the values from the options header and restores them.

Format-compatibility. Our modifications to the IP and transport headers place the encrypted prefix match data back into the same fields as the unencrypted data was originally stored; because comparisons between rules and encrypted data rely on $\leq \geq$, just as unencrypted data, this means that operations performing comparisons on IP and transport headers *remain entirely unchanged at the middlebox*. This ensures backwards compatibility with existing software *and hardware* fast-path operations. Because per-packet operations are tightly optimized in production middleboxes, this compatibility ensures good performance at the cloud despite our changes.

An additional challenge for format compatibility is where to place the decryptable AES data; one option would be to define our own packet format, but this could potentially lead to incompatibilities with existing implementations. By placing it in the IPv6 options header, middleboxes can be configured to ignore this data.[2]

4.1.2 Payload Data

The connection bytestream is encrypted with KeywordMatch. Unlike PrefixMatch, the data in all flows is encrypted with the same key k. The reason is that KeywordMatch is randomized and it does not leak equality patterns across flows.

This allows Embark to support DPI middleboxes, such as intrusion detection or exfiltration prevention. These devices must detect whether or not there exists

[2]It is a common misconception that middleboxes are incompatible with IP options. Commercial middleboxes are usually aware of IP options but many administrators *configure* the devices to filter or drop packets with certain kinds of options enabled.

an exact match for an encrypted rule string *anywhere* in the connection bytestream. Because this encrypted payload data is over the *bytestream*, we need to generate encrypted values which span 'between' packet payloads. Searchable Encryption schemes, which we use for encrypted DPI, require that traffic be *tokenized* and that a set of fixed length substrings of traffic be encrypted along a sliding window – e.g., the word malicious might be tokenized into 'malici', 'alicio', 'liciou', 'icious'. If the term 'malicious' is divided across two packets, we may not be able to tokenize it properly unless we reconstruct the TCP bytestream at the gateway. Hence, if DPI is enabled at the cloud, we do exactly this.

After reconstructing and encrypting the TCP bytestream, the gateway transmits the encrypted bytestream over an 'extension', secondary channel that only those middleboxes which perform DPI operations inspect. This channel is not routed to other middleboxes. We implement this channel as a persistent TCP connection between the gateway and middleboxes. The bytestream in transmission is associated with its flow identifier, so that the DPI middleboxes can distinguish between bytestreams in different flows. DPI middleboxes handle both the packets received from the extension channel as well as the primary channel containing the data packets; we elaborate on this mechanism in [55]. Hence, if an intrusion prevention system finds a signature in the extension channel, it can sever or reset connectivity for the primary channel.

Decryption. The payload data is encrypted with AES and placed back into the packet payload – like PrefixMatch, KeywordMatch is not reversible and we require this data for decryption at the gateway. Because the extension channel is not necessary for decryption, it is not transmitted back to the gateway.

Format-compatibility. To middleboxes which only inspect/modify packet headers, encrypting payloads has no impact. By placing the encrypted bytestreams in the extension channel, the extra traffic can be routed past and ignored by middleboxes which do not need this data.

DPI middleboxes which do inspect payloads must be modified to inspect the extension channel alongside the primary channel, as described in [55]; DPI devices are typically implemented in software and these modifications are both straightforward and introduce limited overhead (as we will see in §6).

4.1.3 HTTP Headers

HTTP Headers are a special case of payload data. Middleboxes such as web proxies do not read arbitrary values from packet payloads: the only values they read are the HTTP headers. They can be categorized as DPI middleboxes since they need to examine the TCP bytesteam. However, due to the limitation of full DPI support, we treat these values specially compared to other payload data: we encrypt the entire (untokenized) HTTP URI using a deterministic form of KeywordMatch.

Normal KeywordMatch permits comparison between encrypted values and rules, but not between one value and another value; deterministic KeywordMatch permits two values to be compared as well. Although this is a weaker security guarantee relative to KeywordMatch, it is necessary to support web caching which requires comparisons between different URIs. The cache hence learns the frequency of different URIs, but cannot immediately learn the URI values. This is the only field which we encrypt in the weaker setting. We place this encrypted value in the extension channel; hence, our HTTP encryption has the same format-compatibility properties as other DPI devices.

Like other DPI tasks, this requires parsing the entire TCP bytestream. However, in some circumstances we can extract and store the HTTP headers statelessly; so long as HTTP pipelining is disabled and packet MTUs are standard-sized (>1KB), the required fields will always appear contiguously within a single packet. Given that SPDY uses persistent connections and pipelined requests, this stateless approach does not apply to SPDY.

Decryption. The packet is decrypted as normal using the data stored in the payload; IP options are removed.

4.2 Rule Encryption

Given a ruleset for a middlebox type, the gateway encrypts this ruleset with either KeywordMatch or PrefixMatch, depending on the encryption scheme used by that middlebox as in Table 1. For example, firewall rules are encrypted using PrefixMatch. As a result of encryption, some rulesets expand and we evaluate in §6 by how much. For example, a firewall rule containing an IP prefix that maps to two encrypted prefixes using PrefixMatch becomes two rules, one for each encrypted prefix. The gateway should generate rules appropriately to account for the fact that a single prefix maps to encrypted prefixes. For example, suppose there is a middlebox that counts the number of connections to a prefix P. P maps to 2 encrypted prefixes P_1 and P_2. If the original middlebox rule is '`if` v `in` P `then counter++`', the gateway should generate '`if` v `in` P_1 `or` v `in` P_2 `then counter++`'.

Rules for firewalls and DPI services come from a variety of sources and can have different policies regarding who is or isn't allowed to know the rules. For example, exfiltration detection rules may include keywords for company products or unreleased projects which the client may wish to keep secret from the cloud provider. On the other hand, many DPI rules are proprietary features of DPI vendors, who may allow the provider to learn the rules, but not the client (gateway). Embark supports three different models for KeywordMatch rules which

allow clients and providers to share rules as they are comfortable: (a) the client knows the rules, and the provider does not; (b) the provider knows the rule, and the client does not; or (c) both parties know the rules. PrefixMatch rules only supports (a) and (c) – the gateway *must* know the rules to perform encryption properly.

If the client is permitted to know the rules, they encrypt them – either generating a KeywordMatch, AES, or PrefixMatch rule – and send them to the cloud provider. If the cloud provider is permitted to know the rules as well, the client will send these encrypted rules annotated with the plaintext; if the cloud provider is not allowed, the client sends only the encrypted rules in random order.

If the client (gateway) is not permitted to know the rules, we must somehow allow the cloud provider to learn the encryption of each rule with the client's key. This is achieved using a classical combination of Yao's garbled circuits [65] with oblivious transfer [40], as originally applied by BlindBox [55]. As in BlindBox, this exchange only succeeds if the rules are signed by a trusted third party (such as McAffee, Symantec, or EmergingThreats) – the cloud provider should not be able to generate their own rules without such a signature as it would allow the cloud provider to read arbitrary data from the clients' traffic. Unlike BlindBox, this rule exchange occurs exactly once – when the gateway initializes the rule. After this setup, all connections from the enterprise are encrypted with the same key at the gateway.

Rule Updates. Rule updates need to be treated carefully for PrefixMatch. Adding a new prefix/range or removing an existing range can affect the encryption of an existing prefix. The reason is that the new prefix can overlap with an existing one. In the worst case, the encryption of all the rules needs to be updated.

The fact that the encryption of old rules changes poses two challenges. The first challenge is the correctness of middlebox state. Consider a NAT with a translation table containing ports and IP addresses for active connections. The encryption of an IP address with EncryptValue depends on the list of prefixes so an IP address might be encrypted differently after the rule update, becoming inconsistent with the NAT table. Thus, the NAT state must also be updated. The second challenge is a race condition: if the middlebox adopts a new ruleset while packets encrypted under the old ruleset are still flowing, these packets can be misclassified.

To maintain a consistent state, the gateway first runs EncryptPrefixes for the new set of prefixes. Then, the gateway announces to the cloud the pending update, and the middleboxes ship their current state to the gateway. The gateway updates this state by producing new encryptions and sends the new state back to the middleboxes. During all this time, the gateway continued to encrypt traffic based on the old prefixes and the middleboxes processed it based on the old rules. Once all middleboxes have the new state, the gateway sends a signal to the cloud that it is about to 'swap in' the new data. The cloud buffers incoming packets after this signal until all ongoing packets in the pipeline finish processing at the cloud. Then, the cloud signals to all middleboxes to 'swap in' the new rules and state; and finally it starts processing new packets. For per-packet consistency defined in [51], the buffering time is bounded by the packet processing time of the pipeline, which is typically hundreds of milliseconds. However, for per-flow consistency, the buffering time is bounded by the lifetime of a flow. Buffering for such a long time is not feasible. In this case, if the cloud has backup middleboxes, we can use the migration avoidance scheme [43] for maintaining consistency. Note that all changes to middleboxes are in the *control plane*.

5 Middleboxes: Design & Implementation

Embark supports the core functionality of a set of middleboxes as listed in Table 1. Table 1 also lists the functionality supported by Embark. In Appendix A, we review the core functionality of each middlebox and explain why the functionality in Table 1 is sufficient to support these middleboxes. In this section, we focus on implementation aspects of the middleboxes.

5.1 Header Middleboxes

Middleboxes which operate on IP and transport headers only include firewalls, NATs, and L3/L4 load balancers. Firewalls are read-only, but NATs and L4 load balancers may rewrite IP addresses or port values. For header middleboxes, per-packet operations remain unchanged for both read and write operations.

For read operations, the firewall receives a set of encrypted rules from the gateway and compares them directly against the encrypted packets just as normal traffic. Because PrefixMatch supports $\leq$ and $\geq$, the firewall may use any of the standard classification algorithms [34].

For write operations, the middleboxes assign values from an address pool; it receives these encrypted pool values from the gateway during the rule generation phase. These encrypted rules are marked with a special suffix reserved for rewritten values. When the gateway receives a packet with such a rewritten value, it restores the plaintext value from the pool rather than decrypting the value from the options header.

Middleboxes can recompute checksums as usual after they write.

5.2 DPI Middleboxes

We modify middleboxes which perform DPI operations as in BlindBox [55]. The middleboxes search through the encrypted extension channel – not the packet payloads themselves – and block or log the connection if a blacklisted term is observed in the extension. Embark

also improves the setup time and security for regular expression rules as discussed in §2.4.

5.3 HTTP Middleboxes

Parental filters and HTTP proxies read the HTTP URI from the extension channel. If the parental filter observes a blacklisted URI, it drops packets that belong to the connection.

The web proxy required the most modification of any middlebox Embark supports; nonetheless, our proxy achieves good performance as we will discuss in §6. The proxy caches HTTP static content (e.g., images) in order to improve client-side performance. When a client opens a new HTTP connection, a typical proxy will capture the client's SYN packet and open a new connection to the client, as if the proxy were the web server. The proxy then opens a second connection in the background to the original web server, as if it were the client. When a client sends a request for new content, if the content is in the proxy's cache, the proxy will serve it from there. Otherwise, the proxy will forward this request to the web server and cache the new content.

The proxy has a map of encrypted file path to encrypted file content. When the proxy accepts a new TCP connection on port 80, the proxy extracts the encrypted URI for that connection from the extension channel and looks it up in the cache. The use of deterministic encryption enables the proxy to use a fast search data structure/index, such as a hash map, unchanged. We have two possible cases: there is a hit or a miss. If there is a cache hit, the proxy sends the encrypted file content from the cache via the existing TCP connection. Even without being able to decrypt IP addresses or ports, the proxy can still accept the connection, as the gateway encrypts/decrypts the header fields transparently. If there is a cache miss, the proxy opens a new connection and forwards the encrypted request to the web server. Recall that the traffic bounces back to gateway before being forwarded to the web server, so that the gateway can decrypt the header fields and payloads. Conversely, the response packets from the web server are encrypted by the gateway and received by the proxy. The proxy then caches and sends the encrypted content back. The content is separated into packets. Packet payloads are encrypted on a per-packet basis. Hence, the gateway can decrypt them correctly.

5.4 Limitations

Embark supports the core functionality of a wide-range of middleboxes, as listed in Table 1, but not all middlebox functionality one could envision outsourcing. We now discuss some examples. First, for intrusion detection, Embark does not support regular expressions that cannot be expanded in a certain number of keyword matches, running arbitrary scripts on the traffic [47], or advanced statistical techniques that correlate different flows studied in the research literature [69].

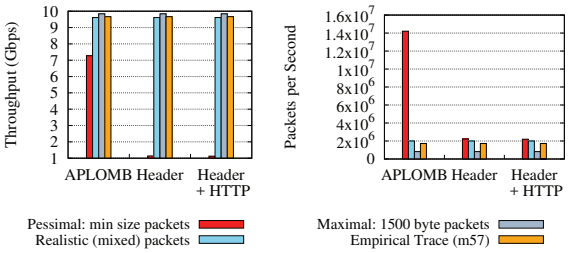


Figure 5: Throughput on a single core at stateless gateway.

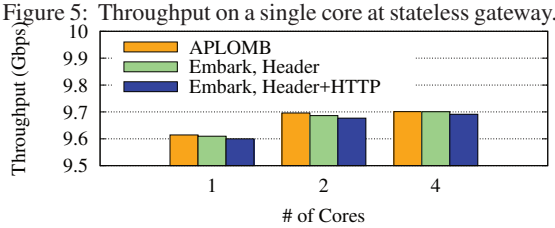


Figure 6: Gateway throughput with increasing parallelism.

Second, Embark does not support application-level middleboxes, such as SMTP firewalls, application-level gateways or transcoders. These middleboxes parse the traffic in an application-specific way – such parsing is not supported by KeywordMatch. Third, Embark does not support port scanning because the encryption of a port depends on the associated IP address. Supporting all these functionalities is part of our future work.

6 Evaluation

We now investigate whether Embark is practical from a performance perspective, looking at the overheads due to encryption and redirection. We built our gateway using BESS (Berkeley Extensible Software Switch, formerly SoftNIC [35]) on an off-the-shelf 16-core server with 2.6GHz Xeon E5-2650 cores and 128GB RAM; the network hardware is a single 10GbE Intel 82599 compatible network card. We deployed our prototype gateway in our research lab and redirected traffic from a 3-server testbed through the gateway; these three client servers had the same hardware specifications as the server we used as our gateway. We deployed our middleboxes on Amazon EC2. For most experiments, we use a synthetic workload generated by the Pktgen [63]; for experiments where an empirical trace is specified we use the m57 patents trace [26] and the ICTF 2010 trace [62], both in IPv4.

Regarding DPI processing which is based on BlindBox, we provide experiment results only for the improvements Embark makes on top of BlindBox, and refer the reader to [55] for detailed DPI performance.

6.1 Enterprise Performance

We first evaluate Embark's overheads at the enterprise.

6.1.1 Gateway

How many servers does a typical enterprise require to outsource traffic to the cloud? Fig. 5 shows the gateway throughput when encrypting traffic to send to the cloud,

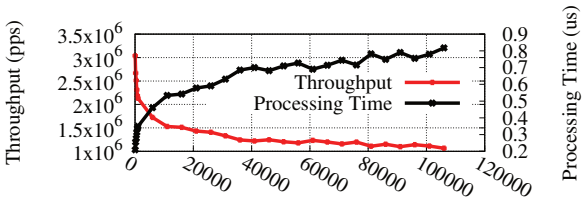


Figure 7: Throughput as # of PrefixMatch rules increases.

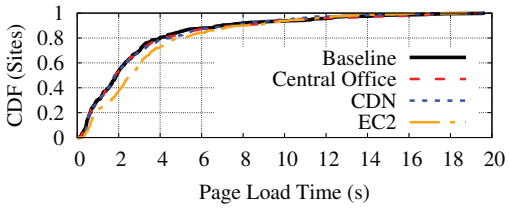


Figure 8: Page load times under different deployments.

first with normal redirection (as used in APLOMB [54]), then with Embark's L3/L4-header encryption, and finally with L3/L4-header encryption as well as stateless HTTP/proxy encryption. For empirical traffic traces with payload encryption (DPI) disabled, Embark averages 9.6Gbps per core; for full-sized packets it achieves over 9.8Gbps. In scalability experiments (Fig. 6) with 4 cores dedicated to processing, our server could forward at up to 9.7Gbps for empirical traffic while encrypting for headers and HTTP traffic. There is little difference between the HTTP overhead and the L3/L4 overhead, as the HTTP encryption only occurs on HTTP requests – a small fraction of packets. With DPI enabled (not shown), throughput dropped to 240Mbps per core, suggesting that an enterprise would need to devote at least 32 cores to the gateway.

How do throughput and latency at the gateway scale with the number of rules for PrefixMatch? In §3.2, we discussed how PrefixMatch stores sorted intervals; every packet encryption requires a binary search of intervals. Hence, as the size of the interval map goes larger, we can expect to require more time to process each packet and throughput to decrease. We measure this effect in Fig. 7. On the y_1 axis, we show the aggregate per packet throughput at the gateway as the number of rules from 0 to 100k. The penalty here is logarithmic, which is the expected performance of the binary search. From 0-10k rules, throughput drops from 3Mpps to 1.5Mpps; after this point the performance penalty of additional rules tapers off. Adding additional 90k rules drops throughput to 1.1Mpps. On the y_2 axis, we measure the processing time per packet, *i.e.*, the amount of time for the gateway to encrypt the packet; the processing time follows the same logarithmic trend.

Is PrefixMatch faster than existing order preserving algorithms? We compare PrefixMatch to BCLO [21] and mOPE [48], two prominent order-preserving encryption schemes. Table 2 shows the results. We can see that PrefixMatch is about four orders of magnitude faster than these schemes.

Operation	BCLO	mOPE	PrefixMatch
Encrypt 10K rules	9333μs	6640μs	0.53μs
Encrypt 100K rules	9333μs	8300μs	0.77μs
Decrypt	169μs	0.128μs	0.128μs

Table 2: PrefixMatch's performance.

What is the memory overhead of PrefixMatch? Storing 10k rules in memory requires 1.6MB, and storing 100k rules in memory requires 28.5MB – using unoptimized C++ objects. This overhead is negligible.

6.1.2 Client Performance

We use web performance to understand end-to-end user experience of Embark. Fig. 8 shows a CDF for the Alexa top-500 sites loaded through our testbed. We compare the baseline (direct download) assuming three different service providers: an ISP hosting services in a Central Office (CO), a Content-Distribution Network, and a traditional cloud provider (EC2). The mean RTTs from the gateway are 60μs, 4ms, and 31ms, respectively. We deployed Embark on EC2 and used this deployment for our experiments, but for the CO and CDN we emulated the deployment with inflated latencies and servers in our testbed. We ran a pipeline of NAT, firewall and proxy (with empty cache) in the experiment. Because of the 'bounce' redirection Embark uses, all page load times increase by some fraction; in the median case this increase is less than 50ms for the ISP/Central Office, 100ms for the CDN, and 720ms using EC2; hence, ISP based deployments will escape human perception [39] but a CDN (or a cloud deployment) may introduce human-noticeable overheads.

6.1.3 Bandwidth Overheads

We evaluate two costs: the increase in bandwidth due to our encryption and metadata, and the increase in bandwidth cost due to 'bounce' redirection.

How much does Embark encryption increase the amount of data sent to the cloud? The gateway inflates the size of traffic due to three encryption costs:

- If the enterprise uses IPv4, there is a 20-byte per-packet cost to convert from IPv4 to IPv6. If the enterprise uses IPv6 by default, there is no such cost.
- If HTTP proxying is enabled, there are on average 132 bytes per request in additional encrypted data.
- If HTTP IDS is enabled, there is at worst a 5× overhead on all HTTP payloads [55].

We used the m57 trace to understand how these overheads would play out in aggregate for an enterprise. On the uplink, from the gateway to the middlebox service provider, traffic would increase by 2.5% due to encryption costs for a header-only gateway. Traffic would increase by 4.3× on the uplink for a gateway that supports DPI middleboxes.

How much does bandwidth increase between the gateway and the cloud from using Embark? How much would this bandwidth increase an enterprises' networking costs? Embark sends all network traffic to and from the middlebox service provider for processing, before

Application	Baseline Throughput	Embark Throughput
IP Firewall	9.8Gbps	9.8Gbps
NAT	3.6Gbps	3.5 Gbps
Load Balancer L4	9.8 Gbps	9.8Gbps
Web Proxy	1.1Gbps	1.1Gbps
IDS	85Mbps	166Mbps [55]

Table 3: Middlebox throughput for an empirical workload.

sending that traffic out to the Internet at large.

In ISP contexts, the clients' middlebox service provider and network connectivity provider are one and the same and one might expect costs for relaying the traffic to and from the middleboxes to be rolled into one service 'package;' given the latency benefits of deployment at central offices (as we saw in Fig. 8) we expect that ISP-based deployments are the best option to deploy Embark.

In the cloud service setting the client must pay a third party ISP to transfer the data to and from the cloud, before paying that ISP a third time to actually transfer the data over the network. Using current US bandwidth pricing [24, 38, 61], we can estimate how much outsourcing would increase overall bandwidth costs. Multi-site enterprises typically provision two kinds of networking costs: Internet access, and intra-domain connectivity. Internet access typically has high bandwidth but a lower SLA; traffic may also be sent over shared Ethernet [24, 61]. Intra-domain connectivity usually has a private, virtual Ethernet link between sites of the company with a high SLA and lower bandwidth. Because bounce redirection is over the 'cheaper' link, the overall impact on bandwidth cost with header-only encryption given public sales numbers is between 15-50%; with DPI encryption, this cost increases to between 30-150%.

6.2 Middleboxes

We now evaluate the overheads at each middlebox.

Is throughput reduced at the middleboxes due to Embark?

Table 3 shows the throughput sustained for the apps we implemented. The IP Firewall, NAT, and Load Balancer are all 'header only' middleboxes; the results shown compare packet processing over the same dataplane, once with encrypted IPv6 data and once with unencrypted IPv4 data. The only middlebox for which any overhead is observable is the NAT – and this is a reduction of only 2.7%.

We re-implemented the Web Proxy and IDS to enable the bytestream aware operations they require over our encrypted data. We compare our Web Proxy implementation with Squid [10] to show Embark can achieve competitive performance. The Web Proxy sustains the same throughput with and without encrypted data, but, as we will present later, does have a higher service time per cache hit. The IDS numbers compare Snort (baseline) to the BlindBox implementation; this is not an apples-to-apples comparison as BlindBox performs mostly exact

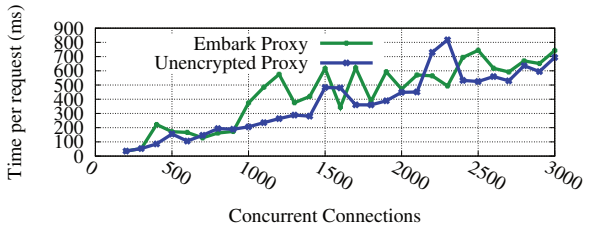


Figure 9: Access time per page against the number of concurrent connections at the proxy.

matches where Snort matches regular expressions.

In what follows, we provide some further middlebox-specific benchmarks for the firewall, proxy, and IDS.

Firewalls: *Does Embark support all rules in a typical firewall configuration? How much does the ruleset "expand" due to encryption?*

We tested our firewall with three rulesets provided to us by a network administrator at our institution and an IP firewall ruleset from Emerging Threats [3]. We were able to encode all rules using range and keyword match encryptions. The size of 3 rulesets did not change after encryption, while the size of the other ruleset from Emerging Threats expanded from 1363 to 1370 – a 0.5% increase. Therefore, we conclude that it has negligible impact on the firewall performance.

Proxy/Caching: The throughput number shown in Table 3 is not the typical metric used to measure proxy performance. A better metric for proxies is how many connections the proxy can handle concurrently, and what time-to-service it offers each client. In Fig. 9, we plot time-to-service against the number of concurrent connections, and see that it is on average higher for Embark than the unencrypted proxy, by tens to hundreds of milliseconds per page. This is not due to computation costs, but instead, due to the fact that the encrypted HTTP header values are transmitted on a different channel than the primary data connection. The Embark proxy needs to synchronize between these two flows; this synchronization cost is what increases the time to service.

Intrusion Detection: Our IDS is based on BlindBox [55]. BlindBox incurs a substantial 'setup cost' every time a client initiates a new connection. With Embark, however, the gateway and the cloud maintain one, long-term persistent connection. Hence, this setup cost is paid once when the gateway is initially configured. Embark also heuristically expands regular expressions in the rulesets into exact match strings. This results in two benefits:

(1) End-to-end performance improvements. Where BlindBox incurs an initial handshake of 97s [55] to open a new connection and generate the encrypted rules, end hosts under Embark never pay this cost. Instead, the gateway pays a one-time setup cost, and end hosts afterwards perform a normal TCP or SSL handshake of only 3-5 RTTs. In our testbed, this amounts to between 30 and 100 ms, depending on the site and protocol – an

improvement of 4 orders of magnitude.

(2) Security improvements. Using IDS rulesets from Snort, we converted regular expressions to exact match strings as discussed in §2.4. In BlindBox, exact match rules can be supported with higher security than regular expressions. With 10G memory, we were able to convert about half of the regular expressions in this ruleset to a finite number of exact match strings; the remainder resulted in too many possible states. We used two rulesets to evaluate this [3, 9]. With the first ruleset BlindBox would resort to a lower security level for 33% of rules, but Embark would only require this for 11.3%. With the second ruleset, BlindBox would use lower security for 58% of rules, but Embark would only do so for 20.2%. At the same time, Embark does not support the lower security level so Embark simply does not support the remaining regexp rules.

It is also worth noting that regular expression expansion in this way makes the one-time setup very slow in one of the three cases: the case when the gateway may not see the rules. The reason is that, in this case, Embark runs the garbled circuit rule-exchange protocol discussed in §4.2, whose slowdown is linear in the number of rules. On one machine, the gateway to server initial setup would take over 3,000 hours to generate the set of encrypted rules due to the large number of keywords. Fortunately, this setup cost is easily parallelizable. Moreover, this setup cost does not occur in the other two rule exchange approaches discussed in §4.2, since they rely only on one AES encryption per keyword rather than a garbled circuit computation which is six orders of magnitude more expensive.

7 Related Work

Middlebox Outsourcing: APLOMB [54] is a practical service for outsourcing enterprise's middleboxes to the cloud, which we discussed in more detail in §2.

Data Confidentiality: Confidentiality of data in the cloud has been widely recognized as an important problem and researchers proposed solutions for software [18], web applications [30, 50], filesystems [19, 36, 31], databases [49, 46], and virtual machines [68]. CryptDB [49] was one of the first practical systems to compute on encrypted data, but its encryption schemes and database system design do not apply to our network setting.

Focusing on traffic processing, the most closely related work to Embark is BlindBox [55], discussed in §2.4. mcTLS [41] proposed a protocol in which client and server can jointly authorize a middlebox to process certain portions of the encrypted traffic. Unlike Embark, the middlebox gains access to *unencrypted data*. A recent paper [67] proposed a system architecture for outsourced middleboxes to specifically perform deep packet inspection over encrypted traffic.

Trace Anonymization and Inference: Some systems which focus on *offline* processing allow some analysis over anonymized data [44, 45]; they are not suitable for online processing as is Embark. Yamada et al [64] show how one can perform some very limited processing on an SSL-encrypted packet by using only the size of data and the timing of packets, however they cannot perform analysis of the contents of connection data.

Encryption Schemes: Embark's PrefixMatch scheme is similar to order preserving encryption schemes [15], but no existing scheme provided both the performance and security properties we required. Order-preserving encryption (OPE) schemes such as [21, 48] are $>$ 10000 times slower than PrefixMatch (§6) and additionally leak the order of the IP addresses encrypted. On the other hand, OPE schemes are more generic and applicable to a wider set of scenarios. PrefixMatch, on the other hand, is designed for our particular scenario.

The encryption scheme of Boneh et al. [22] enables detecting if an encrypted value matches a range and provides a similar security guarantee to PrefixMatch; at the same time, it is orders of magnitude slower than the OPE schemes which are already slower than PrefixMatch.

Acknowledgments

We thank our shepherd, Srinivasan Seshan, and the anonymous reviewers for their thoughtful comments. We're also grateful to Dahlia Malkhi and Ittai Abraham from VMware Research for their valuable feedback on PrefixMatch.

A Sufficient Properties for Middleboxes

In this section, we discuss the core functionality of the IP Firewall, NAT, L3/L4 Load Balancers in Table 1, and why the properties listed in the Column 2 of Table 1 are sufficient for supporting the functionality of those middleboxes. We omit the discussion of other middleboxes in the table since the sufficiency of those properties is obvious. The reason Embark focuses on the core ("textbook") functionality of these middleboxes is that there exist variations and different configurations on these middleboxes and Embark might not support some of them.

A.1 IP Firewall

Firewalls from different vendors may have significantly different configurations and rule organizations, and thus we need to extract a general model of firewalls. We used the model defined in [66], which describes Cisco PIX firewalls and Linux iptables. In this model, the firewall consists of several access control lists (ACLs). Each ACL consists of a list of rules. Rules can be interpreted in the form (*predicate*, *action*), where the *predicate* describes the packets matching this rule and the *action* describes the action performed on the matched packets. The predicate is defined as a combination of ranges of source/destination IP addresses and ports as well as the protocol. The set of possible actions includes "*accept*" and "*deny*".

Let Enc denote a generic encryption protocol, and $(SIP[], DIP[], SP[], DP[], P)$ denote the predicate of a rule. Any packet with a 5-tuple $(SIP, DIP, SP, DP, P) \in (SIP[], DIP[], SP[], DP[], P)$ matches that rule. We encrypt both tuples and rules. The following property of the encryption is sufficient for firewalls.

$$\begin{aligned}&(SIP, DIP, SP, DP, P) \in (SIP[], DIP[], SP[], DP[], P) \Leftrightarrow \\ &\mathsf{Enc}(SIP, DIP, SP, DP, P) \in \\ &\qquad \mathsf{Enc}(SIP[], DIP[], SP[], DP[], P).\end{aligned} \tag{3}$$

A.2 NAT

A typical NAT translates a pair of source IP and port into a pair of external source IP and port (and back), where the external source IP is the external address of the gateway, and the external source port is arbitrarily chosen. Essentially, a NAT maintains a mapping from a pair of source IP and port to an external port. NATs have the following requirements: 1) same pairs should be mapped to the same external source port; 2) different pairs should not be mapped to the same external source port. In order to satisfy them, the following properties are sufficient:

$$\begin{aligned}&(SIP_1, SP_1) = (SIP_2, SP_2) \\ \Rightarrow &\mathsf{Enc}(SIP_1, SP_1) = \mathsf{Enc}(SIP_2, SP_2),\end{aligned} \tag{4}$$

$$\begin{aligned}&\mathsf{Enc}(SIP_1, SP_1) = \mathsf{Enc}(SIP_2, SP_2) \\ \Rightarrow &(SIP_1, SP_1) = (SIP_2, SP_2).\end{aligned} \tag{5}$$

However, we may relax 1) to: the source IP and port pair that belongs to the same 5-tuple should be mapped to the same external port. After relaxing this requirement, the functionality of NAT is still preserved, but the NAT table may get filled up more quickly since the same pair may be mapped to different ports. However, we argue that this expansion is small in practice because an application on a host rarely connects to different hosts or ports using the same source port. The sufficient properties then become:

$$\begin{aligned}&(SIP_1, DIP_1, SP_1, DP_1, P_1) = (SIP_2, DIP_2, SP_2, DP_2, P_2) \\ \Rightarrow &\mathsf{Enc}(SIP_1, SP_1) = \mathsf{Enc}(SIP_2, SP_2)\end{aligned} \tag{6}$$

and

$$\begin{aligned}&\mathsf{Enc}(SIP_1, SP_1) = \mathsf{Enc}(SIP_2, SP_2) \\ \Rightarrow &(SIP_1, SP_1) = (SIP_2, SP_2).\end{aligned} \tag{7}$$

A.3 L3 Load Balancer

L3 Load Balancer maintains a pool of servers. It chooses a server for an incoming packet based on the L3 connection information. A common implementation of L3 Load Balancing uses the ECMP scheme in the switch. It guarantees that packets of the same flow will be forwarded to the same server by hashing the 5-tuple. Therefore, the sufficient property for L3 Load Balancer is:

$$\begin{aligned}&(SIP_1, DIP_1, SP_1, DP_1, P_1) = (SIP_2, DIP_2, SP_2, DP_2, P_2) \Leftrightarrow \\ &\mathsf{Enc}(SIP_1, DIP_1, SP_1, DP_1, P_1) = \\ &\qquad \mathsf{Enc}(SIP_2, DIP_2, SP_2, DP_2, P_2).\end{aligned} \tag{8}$$

A.4 L4 Load Balancer

L4 Load Balancer [4], or TCP Load Balancer also maintains a pool of servers. It acts as a TCP endpoint that accepts the client's connection. After accepting a connection from a client, it connects to one of the server and forwards the bytestreams between client and server. The encryption scheme should make sure that two same 5-tuples have the same encryption. In addition, two different 5-tuple should not have the same encryption, otherwise the L4 Load Balancer cannot distinguish those two flows. Thus, the sufficient property of supporting L4 Load Balancer is:

$$\begin{aligned}&(SIP_1, DIP_1, SP_1, DP_1, P_1) = (SIP_2, DIP_2, SP_2, DP_2, P_2) \Leftrightarrow \\ &\mathsf{Enc}(SIP_1, DIP_1, SP_1, DP_1, P_1) = \\ &\qquad \mathsf{Enc}(SIP_2, DIP_2, SP_2, DP_2, P_2)\end{aligned} \tag{9}$$

B Formal Properties of PrefixMatch

In this section, we show how PrefixMatch supports middleboxes indicated in Table 1. First of all, we formally list the properties that PrefixMatch preserves. As discussed in 3.2, PrefixMatch preserves the functionality of firewalls by guaranteeing Property 3. In addition, PrefixMatch also ensures the following properties:

$$(SIP_1,DIP_1,SP_1,DP_1,P_1) = (SIP_2,DIP_2,SP_2,DP_2,P_2) \Rightarrow \mathsf{Enc}(SIP_1,DIP_1,SP_1,DP_1,P_1) = \mathsf{Enc}(SIP_2,DIP_2,SP_2,DP_2,P_2) \tag{10}$$

The following statements hold with *high probability*:

$$\mathsf{Enc}(SIP_1) = \mathsf{Enc}(SIP_2) \Rightarrow SIP_1 = SIP_2 \tag{11}$$

$$\mathsf{Enc}(DIP_1) = \mathsf{Enc}(DIP_2) \Rightarrow DIP_1 = DIP_2 \tag{12}$$

$$\mathsf{Enc}(SIP_1,SP_1) = \mathsf{Enc}(SIP_2,SP_2) \Rightarrow (SIP_1,SP_1) = (SIP_2,SP_2) \tag{13}$$

$$\mathsf{Enc}(DIP_1,DP_1) = \mathsf{Enc}(DIP_2,DP_2) \Rightarrow (DIP_1,DP_1) = (DIP_2,DP_2) \tag{14}$$

$$\mathsf{Enc}(P_1) = \mathsf{Enc}(P_2) \Rightarrow P_1 = P_2 \tag{15}$$

We discuss how those properties imply all the sufficient properties in §A as follows.

NAT We will show that Eq.(10)-Eq.(15) imply Eq.(6)- Eq.(7). Given $(SIP_1, DIP_1, SP_1, DP_1, P_1) = (SIP_2, DIP_2, SP_2, DP_2, P_2)$, by Eq. (10), we have $\mathsf{Enc}(SIP_1, SP_1) = \mathsf{Enc}(SIP_2, SP_2)$. Hence, Eq.(6) holds. Similarly, given $\mathsf{Enc}(SIP_1, SP_1) = \mathsf{Enc}(SIP_2, SP_2)$, by Eq.(13), we have $(SIP_1,SP_1) = (SIP_2,SP_2)$. Hence, Eq.(7) also holds. Note that if we did not relax the property in Eq.(6), we could not obtain such a proof.

L3 Load Balancer By Eq.(10), the left to right direction of Eq.(8) holds. By Eq.(11)-Eq.(15), the right to left direction of Eq.(8) also holds.

L4 Load Balancer By Eq.(10), the left to right direction of Eq.(9) holds. By Eq.(11)-Eq.(15), the right to left direction of Eq.(9) also holds.

References

[1] Brocade Network Function Virtualization. http://www.brocade.com/en/products-services/software-networking/network-functions-virtualization.html.

[2] Cisco IOS IPv6 Commands. http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6/command/ipv6-cr-book/ipv6-s2.html.

[3] Emerging Threats.net Open rulesets. http://rules.emergingthreats.net/.

[4] HAProxy. http://www.haproxy.org/.

[5] Intel 82599 10 GbE Controller Datasheet. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/82599-10-gbe-controller-datasheet.pdf.

[6] Network Edge Services Products. https://www.juniper.net/us/en/products-services/network-edge-services/.

[7] Network Function Virtualization for Telecom. http://www.dell.com/learn/us/en/04/tme-telecommunications-solutions-telecom-nfv/.

[8] OPNFV: An Open Platform to Accelerate NFV. https://www.opnfv.org/sites/opnfv/files/pages/files/opnfv_whitepaper_103014.pdf.

[9] Snort v2.9 Community Rules. https://www.snort.org/downloads/community/community-rules.tar.gz.

[10] Squid: Optimising Web Delivery. http://www.squid-cache.org/.

[11] Telefónica NFV Reference Lab. http://www.tid.es/long-term-innovation/network-innovation/telefonica-nfv-reference-lab.

[12] What are White Box Switches? https://www.sdxcentral.com/resources/white-box/what-is-white-box-networking/.

[13] ZScaler. http://www.zscaler.com/.

[14] AT&T Domain 2.0 Vision White Paper. https://www.att.com/Common/about_us/pdf/AT&T%20Domain%202.0%20Vision%20White%20Paper.pdf, Nov. 2013.

[15] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 563–574. ACM, 2004.

[16] Ars Technica. AT&T fined $25 million after call center employees stole customers data. http://arstechnica.com/tech-policy/2015/04/att-fined-25-million-after-call-center-employees-stole-customers-data/.

[17] Aryaka. WAN Optimization. http://www.aryaka.com/.

[18] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 267–283. USENIX Association, 2014.

[19] M. Blaze. A Cryptographic File System for UNIX. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 9–16. ACM, 1993.

[20] Bloomberg Business. RadioShack Sells Customer Data After Settling With States. http://www.bloomberg.com/news/articles/2015-05-20/radioshack-receives-approval-to-sell-name-to-standard-general.

[21] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-Preserving Symmetric Encryption. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: The Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 224–241. Springer-Verlag, 2009.

[22] D. Boneh, A. Sahai, and B. Waters. Fully Collusion Resistant Traitor Tracing with Short Ciphertexts and Private Keys. In *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, pages 573–592. Springer-Verlag, 2006.

[23] P. R. Clearinghouse. Chronology of data breaches . http://www.privacyrights.org/data-breach.

[24] Comcast. Small Business Internet. http://business.comcast.com/internet/business-internet/plans-pricing.

[25] I. Cooper, I. Melve, and G. Tomlinson. Internet Web Replication and Caching Taxonomy. IETF RFC 3040, Jan. 2001.

[26] Digital Corpora. m57-Patents Scenario. http://digitalcorpora.org/corpora/scenarios/m57-patents-scenario.

[27] European Telecommunications Standards Institute. NFV Whitepaper. https://portal.etsi.org/nfv/nfv_white_paper.pdf.

[28] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions Using FlowTags. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 533–546. USENIX Association, 2014.

[29] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling Innovation in Network Function Control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 163–174. ACM, 2014.

[30] D. B. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. C. Mitchell, and A. Russo. Hails: Protecting Data Privacy in Untrusted Web Applications. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 47–60. USENIX Association, 2012.

[31] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh. SiRiUS: Securing Remote Untrusted Storage. In *Proceedings of the Tenth Network and Distributed System Security Symposium*, NDSS '03, pages 131–145. Internet Society (ISOC), Feb. 2003.

[32] O. Goldreich. *Foundations of Cryptography: Volume I Basic Tools*. Cambridge University Press, 2001.

[33] M. Goodrich and R. Tamassia. *Introduction to Computer Security*. Pearson, 2010.

[34] P. Gupta and N. McKeown. Algorithms for Packet Classification. *IEEE Network*, 15(2):24–32, Mar. 2001.

[35] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy. SoftNIC: A Software NIC to Augment Hardware. Technical Report UCB/EECS-2015-155, EECS Department, University of California, Berkeley, May 2015.

[36] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, FAST '03, pages 29–42. USENIX Association, 2003.

[37] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. ClickOS and the Art of Network Function Virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 459–473. USENIX Association, 2014.

[38] Megapath. Ethernet Data Plus. http://www.megapath.com/promos/ethernet-dataplus/.

[39] R. B. Miller. Response Time in Man-computer Conversational Transactions. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 267–277. ACM, 1968.

[40] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[41] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 199–212. ACM, 2015.

[42] E. Nordmark. Stateless IP/ICMP Translation Algorithm (SIIT). IETF RFC 2765, Feb. 2000.

[43] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A Framework for NFV Applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 121–136, New York, NY, USA, 2015. ACM.

[44] R. Pang, M. Allman, V. Paxson, and J. Lee. The Devil and Packet Trace Anonymization. *SIGCOMM Computer Communication Review*, 36(1):29–38, Jan. 2006.

[45] R. Pang and V. Paxson. A High-level Programming Environment for Packet Trace Anonymization and Transformation. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 339–351. ACM, 2003.

[46] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind Seer: A Scalable Private DBMS. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 359–374. IEEE Computer Society, 2014.

[47] V. Paxson. Bro: A System for Detecting Network Intruders in Real-time. *Computer Networks*, 31(23-24):2435–2463, Dec. 1999.

[48] R. A. Popa, F. H. Li, and N. Zeldovich. An Ideal-Security Protocol for Order-Preserving Encoding. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 463–477. IEEE Computer Society, 2013.

[49] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100. ACM, 2011.

[50] R. A. Popa, E. Stark, J. Helfer, S. Valdez, N. Zeldovich, M. F. Kaashoek, and H. Balakrishnan. Building Web Applications on Top of Encrypted Data Using Mylar. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 157–172. USENIX Association, 2014.

[51] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for Network Update. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 323–334. ACM, 2012.

[52] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and Implementation of a Consolidated Middlebox Architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 24–24. USENIX Association, 2012.

[53] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi. The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 21:1–21:6. ACM, 2011.

[54] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes

Someone else's Problem: Network Processing As a Cloud Service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 13–24. ACM, 2012.

[55] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 213–226. ACM, 2015.

[56] G. Silowash, T. Lewellen, J. Burns, and D. Costa. Detecting and Preventing Data Exfiltration Through Encrypted Web Sessions via Traffic Inspection. Technical Report CMU/SEI-2013-TN-012, Software Engineering Institute, Carnegie Mellon University, 2013.

[57] P. Srisuresh and K. B. Egevang. Traditional IP Network Address Translator (Traditional NAT). IETF RFC 3022, Jan. 2001.

[58] D. Thaler and C. E. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. IETF RFC 2991, Nov. 2000.

[59] The Snort Project. Snort users manual, 2014. Version 2.9.7.

[60] Verizon. 2015 Data Breach Investigations Report. http://www.verizonenterprise.com/DBIR/2015/.

[61] Verizon. High Speed Internet Packages. http://www.verizon.com/smallbusiness/products/business-internet/broadband-packages/.

[62] G. Vigna. ICTF Data. https://ictf.cs.ucsb.edu/.

[63] K. Wiles. Pktgen. https://pktgen.readthedocs.org/.

[64] A. Yamada, Y. Saitama Miyake, K. Takemori, A. Studer, and A. Perrig. Intrusion Detection for Encrypted Web Accesses. In *21st International Conference on Advanced Information Networking and Applications Workshops*, 2007.

[65] A. C.-C. Yao. How to Generate and Exchange Secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167. IEEE Computer Society, 1986.

[66] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra. FIREMAN: A Toolkit for FIREwall Modeling and ANalysis. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, SP '06, pages 199–213. IEEE Computer Society, 2006.

[67] X. Yuan, X. Wang, J. Lin, and C. Wang. Privacy-preserving Deep Packet Inspection in Outsourced Middleboxes. In *Proceedings of the 2016 IEEE Conference on Computer Communications*, INFOCOM '16, 2016.

[68] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 203–216. ACM, 2011.

[69] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, SSYM'00, pages 13–13. USENIX Association, 2000.