Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Full length article

Oblivious and distributed firewall policies for securing firewalls from malicious attacks

Ali Allami [a,*], Tyler Nicewarner [a], Ken Goss [b], Ashish Kundu [c], Wei Jiang [d], Dan Lin [a]

[a] *Vanderbilt University, Nashville, TN, USA*
[b] *Sandia National Laboratories, Albuquerque, NM, USA*
[c] *Cisco Systems, Inc., San Francisco, CA, USA*
[d] *Oracle Labs, Burlington, MA, USA*

ARTICLE INFO

Keywords:
Privacy-preserving
Secure multiparty computation
Secure firewall

ABSTRACT

Firewalls are effective in preventing attacks initiated from outside of an organization's network, but they are vulnerable to external threats, e.g. ransomware attacks may expose sensitive firewall data to malicious entities or disable network protection from the firewall. In this paper, we present Obliv-FW: a novel distributed architecture and a suite of protocols to obliviously manage and evaluate firewall rules and policies to prevent external attacks oriented to the firewall data. Obliv-FW alleviates this issue by obfuscating the blacklist or whitelist and distributing the function of evaluating these lists across multiple servers residing in different access control zones of the organization's internal network. Thus, both accessing and altering the rules are considerably more difficult thereby providing better protection to the local network as well as greater security for the firewall itself. Obliv-FW is developed by leveraging the existing secure multi-party computation techniques. Our empirical results show that the overhead of Obliv-FW is small, and it can be a very valuable tool to mitigate the ever-increasing threats to a private network from external attacks including ransomware attacks.

1. Introduction

Firewalls are a very common security tool used to protect local networks from threats present in the larger Internet. This is achieved by various strategies, the most basic of which is a packet filter (Peltier and Peltier, 2006). This type of firewall checks each incoming packet against some set of rules often represented by a blacklist or whitelist. With the increasing sophistication of attacks on firewalls from external threats, there has been an interest in distributing the functionality and management of the firewall (Pena and Yu, 2014; Kaur et al., 2015; Satasiya et al., 2016; Filipek and Hudec, 2016). While this does allow a large number of machines in the local network to potentially maintain protection without dependence on the functionality of a central server, it still presents the opportunity for attackers to disable or circumvent the rules of the firewall. Furthermore, if the rules of these firewalls exist in plaintext form, as is often the practice case, the rules themselves may be considered sensitive information due to a desire to hide the identities and locations of external threats.

In addition to the potential threat of ransomware attacks, which can compromise confidential firewall information, an attacker possesses the capability to disable the firewall through the encryption of pertinent firewall-related data and rules. Two distinct scenarios are considered herein for an external attacker.

- In the passive case, an attacker may copy the data contained in the firewall rules. The attacker may learn and share the identity or location information of entities considered as threats to the corporation. The attacker may also learn and leak information regarding allies' or affiliates' identities or locations outside the corporate intranet.
- If instead his attacks are more active, an attacker may maliciously modify firewall rules which could prove disastrous for the network and the organization.

To prevent these attacks, we may argue that firewall rules can be encrypted and access control policies can be used to restrict entities from accessing firewall configuration files. However, these techniques are not sufficient for the following reasons:

- *Encrypted firewall policies and rules*: When the firewall policies and rules are encrypted with a symmetric-key encryption scheme such as AES (NIST, 2001), only the person who has the secret

* Corresponding author.
E-mail addresses: ali.allami@vanderbilt.edu (A. Allami), tyler.j.nicewarner@vanderbilt.edu (T. Nicewarner), kenrgoss@gmail.com (K. Goss), ashkundu@cisco.com (A. Kundu), wei.wj.jiang@oracle.com (W. Jiang), dan.lin@vanderbilt.edu (D. Lin).

https://doi.org/10.1016/j.cose.2024.104201
Received 14 June 2024; Received in revised form 21 October 2024; Accepted 7 November 2024
Available online 16 November 2024

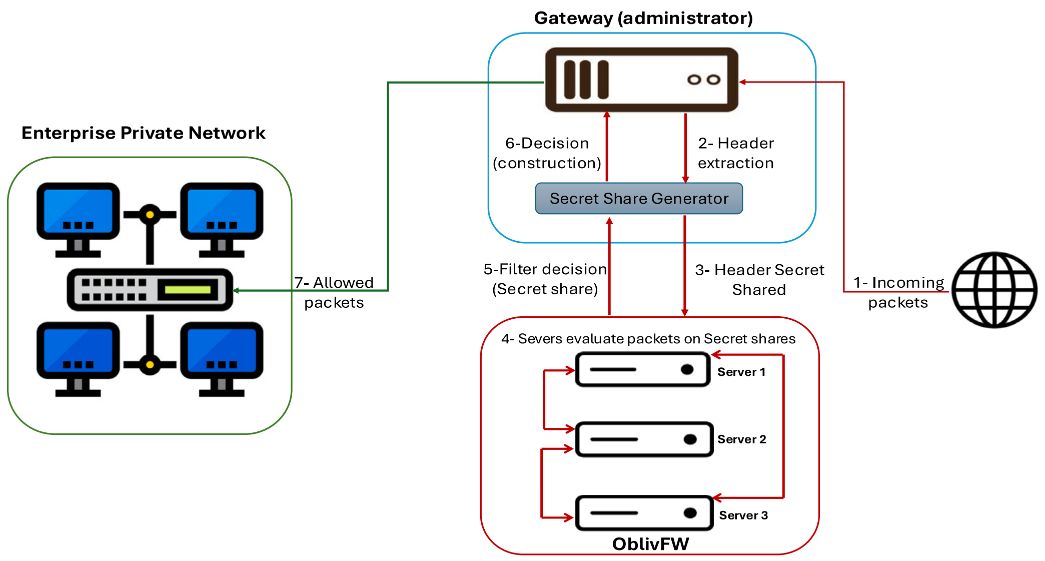

Fig. 1. Obliv-FW system architecture.

key can decrypt these files to access the actual policies and rules. However, during the firewall evaluation process for a new network packet, the rules have to be decrypted first. This opens the door for attackers to access these policies and rules without knowing the secret key. In addition, when a security breach occurs, the secret key may get exposed, and consequently, so do the encrypted firewall files.

- *More restrictive access control policies*: Suppose that mandatory access controls restrict the access to firewall files to the root or a small group of privileged users. Nevertheless, security breaches, exploring unknown vulnerabilities of the network, can bypass access control mechanisms. In addition, human errors and misconfigurations of firewall policies can inadvertently expose sensitive firewall files to other outside attackers.

Existing firewall solutions present in Section 2, while offering protection against various threats, lack comprehensive defense mechanisms against both ransomware attacks and external intrusions. This observation highlights a critical gap in current security paradigms, necessitating the development of novel approaches to firewall management that can effectively address the various nature of modern cyberattacks. Ideally, firewall rules should be encrypted and remain encrypted during the firewall evaluation process. In addition, we should not rely on one server to manage the firewalls because when a security breach occurs, even encrypted data can get disclosed and firewall services can be disabled. Therefore, to hide the firewall rules and policies and achieve a certain degree of fault-tolerance, we need to utilize a distributed architecture and innovative ways to “encrypt” or “hide” the firewall rules and policies. Simultaneously, the architecture and firewall hiding method should allow secure and oblivious firewall policy evaluations without decrypting the hidden firewall information.

1.1. Our contributions

This paper presents a novel distributed firewall architecture that achieves information-theoretic security against local inspection and tampering while incurring minimal performance overhead. By leveraging Bloom filter data structure and secret sharing techniques, our system distributes the firewall rules and functionality, enhancing security compared to existing centralized solutions. It is noteworthy that our proposed system operates on a significantly smaller data domain, requiring only 20 bits compared to existing solutions, which can necessitate data domains of up to 2048 bits in certain scenarios. This reduced data footprint contributes to the efficiency and scalability of our approach. Consequently, integrating existing solutions into our problem setup would likely result in substantial overhead, highlighting the advantages of our system’s optimized design. The proposed architecture offers provable guarantees against attackers, thereby significantly strengthening the resilience of network security infrastructure.

Fig. 1 presents a high-level architectural overview of the proposed system, which necessitates a minimum of three servers. The figure illustrates the data flow for incoming packets from the internet to the enterprise network. It is important to note that the processing of outgoing packets follows a symmetrical architecture, mirroring the flow depicted for incoming packets. our proposed methods will proceed as follows:

(1) System administrators construct and secretly share a Bloom filter to represent firewall rules.
(2) Following the initialization, any gateway server receiving a packet invokes a set membership query concerning the related information.
(3) The distributed firewall servers, acting as shareholders in a secret sharing scheme, cooperate to derive shares of the result, sent to the coordinating gateway.
(4) The coordinating gateway reconstructs the result of the distributed computation and either rejects or forwards the packet to the internal network.

Consider a comparison with a standard firewall, even if it is distributed across multiple machines and an attacker has access to one of these machines, the attacker can learn firewall information stored on that machine and corrupt its content. Because of the way the firewall is distributed in our scheme, accessing the firewall data on one or more (up to a predefined threshold) servers, no information can be leaked regarding the firewall. Even when the firewall data is modified by a malicious attacker, our firewall can still function properly. To summarize, our contribution comes in several folds as follows:

(1) We introduce a novel distributed firewall architecture that leverages either additive or Shamir secret sharing schemes (Shamir, 1979) to distribute firewall rules securely among participating servers, preventing any single server from reconstructing the complete configuration and mitigating local inspection risks. Unlike previous approaches, our system integrates robust evaluation mechanisms, effectively preventing attackers who may have compromised individual servers and preventing them from bypassing the firewall’s security measures.

(2) Our distributed firewall architecture exhibits resilience to server failures, leveraging the inherent properties of Shamir's secret sharing scheme [Shamir, 1979]. This enables the firewall to remain operational as long as a predefined threshold (t) of the m servers remains active. This resilience to component failure is a key strength of our design. Section 4 details the implementation of these resilience properties, incorporating novel strategies alongside existing research on Shamir's secret sharing.
(3) We employ secure multi-party computation (SMC) techniques to securely and obliviously evaluate firewall criteria using distributed and secret shared Bloom filters. These filters, constructed from desired firewall functionalities, can be easily integrated into existing routing and firewall protocols with minimal software modification. While this approach introduces a non-negligible latency cost, it offers enhanced security, demonstrating the inherent trade-off between performance and security. Our experimental setup provides empirical estimates of this cost.
(4) We provide an analysis of the security and efficiency of our proposed scheme. We demonstrate its resilience against previously mentioned attacks and analyze the complexity of our approaches. Additionally, we present a range of options for modifying the system to balance security and efficiency trade-offs.

1.2. Threat model and assumptions

Here we classify the participating entities into parties (servers) and administrators inspired by the Embark architecture (Lan et al., 2016) and specify their allowable behaviors. In addition, we define our assumption regarding the initial state of the system and the trust model. The following notations will be used throughout the paper:

- $P_1, \ldots, P_m$: a set of m servers that obliviously store and manage our distributed firewalls where $m \geq 3$. The setup of a multi-server framework is aligned with the General Data Protection Regulation (GDPR), the toughest privacy and security law passed by the European Union (GDPR.eu, 2023).
- A: The system administrator, possessing the exclusive authority to initialize the firewall rules, performs a secret sharing operation to distribute these rules among the firewall servers. This operation ensures that the administrator retains no access to t firewall servers at the same time, effectively separating the initialization process from the operational environment.

To access P_i, it requires permissions from at least t system administrators, where t is a pre-defined threshold. When an attacker compromises P_i, we classify the attacker as either passive or active:

- *Passive adversary*: An attacker may access one or more P_i servers and simply read the firewall-related files and observe LAN traffic passing through the servers.
- *Active adversary*: An attacker may access one or more P_i servers and modify the firewall-related data and LAN traffic through these servers. The attacker may also disable these servers.

Under the proposed system architecture, we develop two classes of secure protocols to obliviously manage firewalls. The first class offers security guarantees against passive adversaries, and the other class can detect malicious activities, identify the servers involved in these activities, and recover from their activities up to a threshold.

1.3. Organization

This paper is organized as follows: Section 2 reviews relevant background on firewalls and recent developments. Section 3 introduces the foundational tools (Bloom Filters and Secret Sharing) used in our scheme. Section 4 presents our protocols in detail, including functionality, procedure, and analysis of complexity, security, and correctness. Section 5 provides empirical results demonstrating the feasibility of the proposed protocols. Finally, Section 7 summarizes our contributions and outlines future research directions.

2. Related work

Firewalls in general have been in existence nearly as long as computer networking. The earliest functions simply involved a router or gateway server maintaining a list of addresses to block or permit. These are so-called *blacklists* and *whitelists* respectively (Peltier and Peltier, 2006). In the whitelisting, the default policy is to deny all traffic flows. Users can then selectively grant permission for specific traffic flows, thus establishing a whitelist. In contrast, the blacklisting operates under the inverse principle. By default, all traffic flows are permitted. Users can then selectively deny specific traffic flows, effectively creating a blacklist.

As sophistication increased, divergent interests produced multiple approaches to securing local subnets against threats from the Internet at large. These include rules to block entire domains or sub-domains, as well as a wide array of means by which these rules may be optimized for both security and efficiency Gouda and Liu (2004). While very quick, this method potentially allows for a single point of failure in terms of both network functionality, as well as security. It is possible in this case that an adversary may be able to access the list of permitted or prohibited addresses and use this information outside the desired context for nefarious purposes. Additionally, if access to the file containing the lists is obtained, an adversary can grant access to other external undesirable and previously blocked servers, or hinder the ability of a legitimate server to connect. These are important and perhaps critical concerns depending on the setting and desire for security.

One means by which a firewall may be represented in terms of a blacklist or whitelist is through the structure of a Bloom Filter (Geravand and Ahmadi, 2013) which can efficiently give set membership in probabilistic terms. The efficiency of this scheme is particularly strong concerning space. This important topic, on which our work is dependent, will be addressed more thoroughly in Section 3. This data structure has been used in firewall constructions previously to good effect (Tarkoma et al., 2012).

A considerable amount of research has been conducted in reliably distributing the functionality of a firewall throughout a network with varying rates of efficiency and security (Broder and Mitzenmacher, 2004). This is motivated by a variety of concerns for an array of settings from cellular networks to cryptocurrencies and enterprise systems (Geravand and Ahmadi, 2013; Einziger and Friedman, 2017). While advantageous from a resilience perspective, in a situation where restricted network topology is not possible or practical, this approach can have risks of its own (Ioannidis et al., 2000). If an attack is raised in a network and an attacker gains access to the firewall for a particular device, the attacker may take steps to circumvent this policy. This is much more readily achievable for an attacker who gains direct access to the system implementing the firewall than if it were a separate dedicated and more secure system in the network infrastructure for the subnet. Our interest in distributing the firewall is to distribute it in an oblivious means, distinct from these approaches, such that the individual machines cannot simultaneously alter the result of the firewall and maintain anonymity. This is discussed in greater depth in our proposed protocols given in Section 4.

A combined approach allows for the distribution of the firewall, represented by a set of Bloom filters (Maccari et al., 2007). This is useful especially when network topology is not constrained to a single gateway making the bridge between the local network and the Internet, a common occurrence in mesh networks. In this setting, the filters are individually based on the acceptance of packets at each node (Maccari et al., 2007; Neira et al., 2008), and forwarding a packet from one neighbor to another is dependent on the filters associated with the forwarding table. A packet is forwarded if the destination node has whitelisted the source node. This approach still allows for potential manipulation of the filters without the ability to detect malicious activities, and it is less efficient than the centralized approaches.

Hardware-defined networking has proven to be a versatile platform for the execution of a diverse range of network functions, including firewalling, through specialized hardware acceleration. ShieldBox (Trach et al., 2018) addresses the secure deployment of high-performance network functions (NFs) on untrusted servers. It leverages Intel SGX to protect both middlebox execution and data, providing a developer-friendly Click-based interface. ShieldBox integrates Intel DPDK for efficient packet processing and offers features like secure packet processing elements, shared memory packet transfer for NFV chaining, and secure state persistence. While achieving near-native performance, ShieldBox currently focuses on L2 and L3 network processing.

SafeBricks (Poddar et al., 2018), LightBox (Duan et al., 2019), and EVE (Han et al., 2020) represent frameworks that leverage Intel SGX to enhance the security of network functions (NFs) in the cloud. SafeBricks focuses on secure NF outsourcing, enforcing the least privilege across a chain, and utilizing Rust for safety and isolation. LightBox enables secure and efficient execution of stateful middleboxes by introducing a virtual network interface for secure packet access and a state management module for efficient handling of large state footprints. EVE, on the other hand, provides a complete platform for building secure middleboxes with visibility over encrypted traffic, offering high-level Rust APIs for ease of development and enhanced security. EVE supports diverse use cases and multiple encryption protocols, including TLS, DTLS, VPN tunnels, and nested encryption, while ensuring memory safety through Rust and SGXBounds. All three frameworks achieve significant performance gains but inherit limitations from Intel SGX, including limited enclave memory and vulnerability to side-channel attacks.

Software-defined networking has recently shown great promise to ameliorate the issues related to much of networking's inherent hardware dependence. It allows for both hardware abstraction, leading to greater flexibility, as well as a separation between data handling and forwarding and traffic controls (Kreutz et al., 2015). This separation is certainly advantageous from the perspective of flexibility and efficiency, as witnessed by more than a decade of ongoing effort to update the infrastructure from IPv4 to IPv6 so the connection is both feasible and reliable (Internet Society, 2018). Even today, adoption is estimated at around only 23% (Google, 2017). While this has many advantageous properties, it does not inherently impart any greater security to the operations which is a primary motivator for our work. Firewalls implemented under this paradigm may still leak undesirable information, and they may still be maliciously altered.

Khakpour and Liu's work (Khakpour and Liu, 2012) introduced Ladon, a framework for private firewall outsourcing to the cloud. Ladon anonymizes firewall policies using Bloom filters to prevent direct access by the cloud provider. However, Ladon lacks provable security guarantees and remains vulnerable to traffic analysis attacks, allowing potential inference of sensitive policies. Curtmola et al. (2006) proposed a dynamic SSE scheme with optimal search complexity, suggesting its potential for network function outsourcing.

Building upon BlindBox (Sherry et al., 2015), Embark (Lan et al., 2016) introduced the PrefixMatch encryption scheme for efficient handling of packet headers for firewall outsourcing. Embark still necessitates server-side modifications and lacks support for incremental rule updates. Additionally, Embark incurs significant overhead for rule updates, requiring reconstruction of the entire encrypted rule set with each change.

The zero-knowledge middlebox (ZKMB) paradigm, introduced by Grubbs et al. (2022), leverages zero-knowledge proofs to enable clients to demonstrate policy compliance without revealing traffic details beyond policy adherence, offering strong privacy guarantees and protocol compatibility. However, the initial ZKMB prototype suffered from performance limitations. Zombie (Zhang et al., 2023) addressed these limitations by incorporating techniques such as precomputation, asynchrony, and batching to significantly reduce client and middlebox overhead. While ZKMBs offer a promising approach for balancing network policy enforcement with user privacy, current implementations face limitations in both performance and functionality. Computational overhead for both clients and middleboxes remains a significant challenge, as generating and verifying zero-knowledge proofs can be computationally expensive, leading to latency in network traffic. Additionally, the size of zero-knowledge proofs contributes to increased bandwidth consumption, and both clients and middleboxes require substantial memory resources to participate.

Table 1
An example of whitelist firewall policies (Kurose and Ross, 2017).

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	—
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	—
deny	all	all	all	all	all	all

3. Preliminaries

For our proposed methods to follow, we require two additional pieces of research upon which our solution is built. These are Bloom Filters, as previously mentioned, as well as secret sharing.

3.1. *Firewall rules classification*

To follow one of the common practices, we adopt the blacklist/whitelist configuration approach to represent firewall rules or policies. An example of a firewall represented using a whitelist configuration is given in Table 1. It is worth noting that the first 4 rows represent the allowed traffic (whitelist). The last row shows that the rest of the traffic is blocked. We classify the rules into two main categories:

- Type A: simple rules that do not have the comparison operator. For example, if we remove the ">" operator from the "source port" column of the first rule in Table 1, the rule becomes a simple rule since its hash can be directly computed and represented using a Bloom filter.
- Type B: rules that contain the comparison operator. E.g., the first rule of Table 1. Representing these rules using the Bloom filter is not straightforward. However, if the range of the ports is small, a type B rule can be represented by a set of type A rules which can then be represented using the Bloom filter.

These types of rules are very common in practice, and in the paper, the proposed protocols can handle both.

4. The proposed protocols

In this section, we propose our solutions by which a firewall can be secured in an information-theoretic sense and distributed across multiple servers that evaluate the firewall functions quickly and efficiently while maintaining a higher level of security than previously proposed systems. The proposed solutions can also be expanded to include resilience in the face of some subset of servers failing and being additionally secure against attempts to manipulate the firewall via the

Table 2
Common notations.

η	The number of rules to add to the Bloom filter
β	Number of bit locations in the Bloom filter (Bloom filter size)
κ	Number of hash functions used in checking set membership via the Bloom filter
t	Threshold from the Shamir secret sharing scheme
m	Number of firewall management servers
N	Prime modulus for the secret sharing scheme

primitives and nature of the underlying mathematical constructs. To begin with, we focus on simple firewall rules or type A rules defined in Section 3.1. Later, we will discuss how to handle type B rules. The protocol includes three components according to different aspects of firewall management: (1) firewall initialization, (2) firewall rule evaluation, and (3) firewall rule or policy update.

- *Firewall initialization*: Given a false positive error rate, the size of a list of firewall rules, the initialization protocols can decide the Bloom filter size and the number of hash functions. Then the Boom filter representation of the list of firewall rules will be produced. Depending on the adversary model, either additive or Shamir secret sharing scheme, as discussed in Appendix B, will be used to generate a secretly shared Bloom filter, and each share is provided to one of the designated servers for oblivious firewall management and evaluation. Thus, each server has shares of the Bloom filter representing the rules of the firewall, and they are collectively ready for firewall rule evaluations. This initialization of generating secret shares of the Boom filter is done by the administrators who have access to the servers. After that, the original firewall rule list can be discarded for extra security.
- *Firewall rule evaluation*: We developed several protocols for firewall rule evaluation with various security guarantees. In the proposed protocols, we assume that the firewall has been initialized.
- *Firewall rule or policy update*: When firewall rules and policies are updated, the Bloom filter representing this firewall also needs to be updated accordingly. In addition, secret shares of the Bloom filter need to be updated as well on each server. We present a way to perform these actions to update our oblivious firewall.

The proposed schemes are described with several key system parameters shown in Table 2.

4.1. Initialization

The main challenge during the firewall initialization process is to determine the Bloom filter size and the number of hash functions needed to construct the filter. Eqs. (1) and (2) can help the decision-making in this regard. Since the size of the firewall list, η is known and the probability of a false positive is approximated by $0.6185^{\beta/\eta}$, fixing the false positive rate will lead to the size of the Bloom filter β. Consequently, the number of hash functions κ can be calculated by Eq. (2). For example, suppose η= 1,000,000 (one million), and the false positive rate is 0.001. Then $\beta \approx 14.5$ million bits. As a result, the number of hash functions is about $14.5 \times \ln 2 \approx 10$. As for the hash functions, we can adopt SipHash with a set of keys of size κ indexed by t. Our setting is exactly the type of setting for which SipHash was designed and it performs very well (Aumasson and Bernstein, 2012).

$$p = \left(1 - \frac{1}{\beta}\right)^{\kappa\eta} \approx e^{-\kappa\eta/\beta} \tag{1}$$

$$\kappa \approx \frac{\beta}{\eta} \ln 2 \tag{2}$$

During a trusted initialization phase, the addresses forming the firewall list are hashed with the prescribed functions, and the Bloom filter is constructed. This vector of bits is secretly shared using the Gen_Shares function that can be implemented using methods introduced in Appendix B. The shares are sent to each of the shareholder servers to participate in the scheme. Once these two steps have been completed, the online operation may commence. The main steps of the initialization process are presented in Algorithm 1.

Algorithm 1: firewallInit($\langle \text{Admin}, R_s\rangle, \langle P_i, \perp\rangle$) $\rightarrow$ ($\langle \text{Admin}, \perp\rangle, \langle P_i, [\phi_j]_N^{P_i}\rangle$)

```
Input: Firewall rules $R_s$, $1 \le s \le \eta$
Output: $\phi_j$ for $P_i$, $0 \le j < \beta$ and $1 \le i \le m$
1 Admin
    (a) $\phi_j \leftarrow 0$, for $0 \le j < \beta$
    (b) For $1 \le s \le \eta$ do
            $\phi_j \leftarrow 1, \forall j \in \{hash_t(R_s) \mod \beta$, for $1 \le t \le \kappa\}$
    (c) Gen_Shares($\phi_j$), for $0 \le j < \beta$
    (d) Send $[\phi_j]_N^{P_i}$ to $P_i$, for $0 \le j < \beta$ and $1 \le i \le m$
```

4.2. Secure and distributed firewall evaluation against passive adversaries

As mentioned before, we developed two secure protocols to evaluate firewall rules obliviously distributed among m servers: $P_1, \ldots, P_m$. In this section, we discuss the first protocol when the adversaries are passive. The second proposed protocol presented in Section 4.3 is secure against malicious attacks, and it can also mitigate errors introduced by the malfunction of the network components. The main evaluation condition behind the first protocol is based on the following conditions determined by the construction of the Bloom filter:

- Let γ represent a packet header using the firewall rule format given in Table 1. It is extracted by one of the servers $P_1, \ldots, P_m$ from the incoming message. Here we focus on the distributed firewall management setting where each server is responsible for handling a subset of incoming messages. The server that receives an initial packet is called the gateway. It subsequently initiates the firewall evaluation protocols.
- γ of the incoming message matches one of the firewall rules if the sum of the values stored in the Bloom filter locations indexed by the κ hash functions with γ as their input is equal to κ.

If the above condition is true, the packet will be blocked. The design of our first protocol is to ensure that the condition can be verified without disclosing the actual content of the Bloom filter. The steps of the protocol are presented in Algorithm 2.

4.2.1. Key steps and correctness analysis

Note that the parameter N denotes the size of the secret shares. In this protocol, to represent the sum of the values from locations indexed by the κ hash functions, N needs to be bigger than κ. In step 1, the gateway receives a packet from which γ is extracted and sent to the other servers implementing the distributed secret sharing scheme. In step two, the servers hash the received γ with all κ hash functions modulo β, and this set of calculated values are the indices to the locations of the secretly shared Bloom filter, and the values (i.e., secret shares) stored in these locations should be summed. This summation is computed, and the result is sent back to the Gateway. In step 3, the Gateway reconstructs the result of the computation and compares it against κ. Due to the nature of the Bloom filter, this result will be equal to κ if and only if every location in the summation was 1, which is exactly the criteria indicating set membership according to the Bloom filter construction.

Algorithm 2: firewallEval($\langle \text{Gateway}, \gamma \rangle, \langle P_i, [\phi_j]_N^{P_i} \rangle$) $\rightarrow$ ($\langle \text{Gateway}, \sigma \rangle, \langle P_i, \perp \rangle$)

```
Input: Each server $P_i$ has shares $[\phi_j]_N^{P_i}$ of the Bloom filter, for
       $1 \le i \le m$ and $0 \le j < \beta$
Output: $\sigma = \kappa$: the Gateway blocks the packet
1 Gateway
    (a) Derive $\gamma$ from the incoming message
    (b) Send $\gamma$ to $P_i$, for $1 \le i \le m$
2 For each $P_i$:
    (a) $[\sigma]_N^{P_i} \leftarrow \sum_j [\phi_j]_N^{P_i} : j \in \{hash_t(\gamma) \mod \beta, \text{ for } 1 \le t \le \kappa\}$
    (b) Send $[\sigma]_N^{P_i}$ to Gateway
3 Gateway
    (a) $\sigma \leftarrow \text{rebuild}\left([\sigma]_N^{P_1}, \dots, [\sigma]_N^{P_m}\right)$
    (b) If $\sigma = \kappa$: block packet
        else: forward packet
```

4.2.2. Security analysis under passive attacks

We define our notions of security in consonance with the proposed definitions and conventions of Goldreich (2004). As discussed at length, a protocol is secure in the computation of a desired functionality given that the view of the execution of the protocol is indistinguishable from that generated by a simulator. Here we define our security in information-theoretic terms for passive attacks as described previously. These attacks include attempts to extract information from what data may be visible on a compromised server and observing traffic on a compromised server, discussed in Section 1.2.

When an adversary compromises P_i, P_i becomes an alias for the adversary. Therefore, in the subsequent analysis, we simplify the notations by using P_i to represent the adversary who controls the server P_i. Since the Bloom filter is secretly shared among $P_1, \dots, P_m$ and the result sent from P_i to the gateway is also in secret share form, P_i will not be able to learn anything regarding the actual firewall. Next, we present a formal security analysis based on Goldreich (2004).

Given a functionality $f(x, y)$ operating on the inputs x and y and a protocol π implementing this functionality, then an execution image for π is denoted by $\{\text{VIEW}_1^{\pi}(x, y)\}_{x,y\in\{0,1\}^*}$ for server P_1 (as an example). What is needed to prove the security of π is an algorithm S_1 which, given the public information as well as the necessary private information from P_1, P_1 can produce a simulated execution image of π, denoted by $\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*}$ for some simulator algorithm S_1. If the simulated and real execution images are equivalent in an information-theoretic sense, then the protocol is secure with this guarantee for passive adversaries. Thus, the goal is to demonstrate the following equivalence which is given for a server P_1 though, in general, it would need to be demonstrated for all servers:

$$\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*} \equiv \{\text{VIEW}_1^{\pi}(x, y)\}_{x,y\in\{0,1\}^*}$$

Since our protocol is symmetric; that is, all servers or servers $P_1, \dots, P_m$ who are responsible for obliviously managing the firewalls to perform the same operations, it is sufficient to show the equivalence from one server's perspective (e.g., P_1) to prove the protocol is secure for all servers.

The view of the servers in this situation is comprised of two different components. First, they receive shares of the Bloom filter, these shares disclose no information regarding the filter itself, or the firewall rules it represents. This is immediate from the underlying secret sharing scheme. They additionally receive IP addresses that are queried for membership in the set comprising the firewall rules. The only use of this information is in generating the indices for share summation.[1] Summing the shares, or any other local operation performed, yields no information to a shareholder without bounds on available computational power. This too is immediate from the properties of the secret sharing scheme. Thus, a trivially implementable simulator for this view consists of $(\beta + 1)\log_2 N$-bit randomly generated messages. That is, the first β messages simulate the received secretly shared Bloom filter and the last message simulates the summation result. Since this simulation is indistinguishable from the actual protocol execution, the servers' calculations reveal no information among themselves. This is the property we wished to demonstrate for passive adversaries.

4.2.3. Protocol implementation issues

The protocol given in Algorithm 2 can be implemented using either additive or Shamir secret sharing schemes. Under the additive scheme, the Gateway has to collect all the shares before constructing σ. As a result, when a share is lost, the protocol may not proceed properly. Under Shamir, the Gateway only needs to collect t out of m shares to derive σ. If some degree of fault tolerance (regarding packet loss) is desired, we can implement the protocol using the Shamir secret sharing scheme. However, this will incur more computation costs. The empirical results presented in Section 5 will show the performance differences between the two schemes.

4.3. Secure and distributed firewall evaluation against active adversaries

To prevent negative impacts caused by active adversaries, we discuss the changes required to the previous protocol. There are two main modifications: (1) the protocol has to use Shamir's secret sharing scheme, and (2) the consistency of the shares needs to be verified before accepting the result at step 3 of Algorithm 2. The first modification is straightforward, and we just need to share the values using the Shamir scheme and its related operations. Next, we detail how to verify the consistency of the shares.

4.3.1. Verifying share consistency

Depending on the threshold t and m under Shamir's secret sharing, we can employ different strategies to either verify the consistency of the shares or eliminate the negative impact of the inconsistent shares.

4.3.1.1. ***Approach 1 – Berlekamp–Welch error correcting*** When $t < \frac{m}{3}$, we base the additional steps on the well-known, resilient, and efficient Berlekamp–Welch algorithm for error correction (Welch and Berlekamp, 1986). Other algorithms with additional beneficial properties exist such as the various algorithms useful in list decoding (Cohn and Heninger, 2013; Guruswami and Sudan, 1998). Since these return a list of possible polynomials and we lack any means of distinguishing correct from incorrect possibilities, the gain by using such methods is negated for our application, and they simply induce additional computational cost. At step 3 of Algorithm 2, the gateway server already has shares of a polynomial, required by the Berlekamp–Welch algorithm. Rather than rebuilding the shares to reclaim the secret as is normally done, the shares received are used to construct a linear system that is solved locally. This allows for the correct reconstruction of the polynomial underlying the shares representing the filter evaluation, which is equivalent to revealing the shared secret, but with some extra benefit due to special properties of the system. The algorithm works as follows:

- Suppose that the shares $\epsilon_1, \dots, \epsilon_{t'}$ are maliciously modified by t' servers, and $\mathcal{E}(x) = (x - \epsilon_1) \cdots (x - \epsilon_{t'})$, where $t + 2t' + 1 = m$. In addition, let $\mathcal{P}(x)$ be the polynomial that secretly shares m. Define $\mathcal{Q}(x) = \mathcal{P}(x)\mathcal{E}(x)$, which is a polynomial of degree $t + t'$.

[1] These addresses are public information for the servers, so we do not need to produce simulated messages for them.

  - $\mathcal{Q}(x) = a_{t+t'}x^{t+t'} + \cdots + a_1x + a_0$
  - $\mathcal{E}(x) = x^{t'} + b_{t'-1}x^{t'-1} + \cdots + b_1x + b_0$

- Since $\mathcal{Q}(i) = [\sigma]_N^{P_i}\mathcal{E}(i)$, from the m shares $[\sigma]_N^{P_1}, \dots, [\sigma]_N^{P_m}$, a system of m linear equations can be established with m unknowns: $a_{t+t'}, \dots, a_0$ and $b_{t'-1}, \dots, b_0$. Any missing shares (up to t') can be replaced with a value from the domain of these shares.
- Solving the m unknowns leads to $\mathcal{Q}(x)$ and $\mathcal{E}(x)$, from which $\mathcal{P}(x)$ is obtained by $\frac{\mathcal{Q}(x)}{\mathcal{E}(x)}$.

In our description of the Berlekamp and Welch algorithm, $t + 2t' + 1 = m$ where t is the degree of the polynomial and t' is the total errors (caused by active adversaries or network faults) that can be corrected. Since t also represents the number of malicious adversaries or the maximum number of servers that can be compromised, we can set $t = t'$. Therefore, $3t + 1 = m$ implies that $t < \frac{m}{3}$. At the end, the gateway can obtain σ by evaluating $\mathcal{P}(0)$.

4.3.1.2. ***Approach 2 – Enumerating share combinations*** When $\frac{m}{3} \leq t$ and $2t \leq m - 2$, we can enumerate all possible share combinations to reconstruct σ. The gateway server performs the following steps:

- Select one of $\binom{m}{t+2}$ share combinations from the m received shares $[\sigma]_N^{P_1}, \dots, [\sigma]_N^{P_m}$.
- Based on previously chosen $t+2$ shares, derive a polynomial $\mathcal{P}(x)$.
  - If $\mathcal{P}(x)$ has a degree of t, then set $\sigma = \mathcal{P}(0)$ and stop the computation.
  - Otherwise, go to the first step and repeat.
- If any one of the $\binom{m}{t+2}$ combinations returns an invalid $\mathcal{P}(x)$, the gateway will alarm the system admins about potential security breaches occurring on the firewall servers.

The reason we choose $t+2$ shares to reconstruct $\mathcal{P}(x)$ is that if all $t+2$ shares are valid, the resulting polynomial must have a degree of t since $[\sigma]_N^{P_1}, \dots, [\sigma]_N^{P_m}$ were produced from t-degree polynomials. It is a well-known fact that if one share is invalid, $\mathcal{P}(x)$ should have a degree of $t+1$ with overwhelming probability (determined by the size of N).

4.3.1.3. ***Approach 3 – Only detecting share inconsistencies*** When $t \leq m - 2$, the gateway server derives a polynomial $\mathcal{P}(x)$ using all m shares.

- If $\mathcal{P}(x)$ has a degree of t, then set $\sigma = \mathcal{P}(0)$.
- Otherwise, the gateway alarms the system admins about potential security breaches occurring on the firewall servers.

The approach follows the same logic as discussed in the second approach related to share verification.

4.4. Firewall updates

In many situations, after the establishment of security measures, new threats may be identified and added to the list of firewall rules. If the list structure were static, this could pose some problems since a new Bloom filter would have to be constructed and shared from scratch. Thus, we propose a simple means by which elements may be obliviously added to the Bloom filter representing the firewall rules without disturbing elements already encoded in that structure.

The following computation must be executed for the new rule R to be inserted in the filter.

$$[\phi_j]_N^{P_i} = [1]_N^{P_i} : j \in \{hash_t(R) \mod \beta, 1 \leq t \leq \kappa\}$$

This operation is invoked by the administrators who can collaboratively access server P_i. Care must be taken in this operation since the shared values in the Bloom filter should be exclusively 0 or 1, and we cannot allow this manipulation to introduce shares of elements aside from those two. Thus, we cannot simply increment the shares for each location resulting from the modular hash of the address in question. In the above expression, each location, if it is previously 0, gets incremented to 1. Additionally, if it is previously 1, it remains unchanged, as desired.

4.5. Alternative design: Firewall evaluation without using bloom filter

Our firewall evaluation protocol leverages Bloom filters, offering efficiency with a potential for false positives (FPs). We can mitigate this by increasing the number of hash functions, as detailed in Section 5.5. Alternatively, Algorithm 3 directly utilizes blacklists or whitelists, eliminating FPs. Given the limited size of practical firewall rule sets (Al-Shaer and Hamed, 2004; Hamed and Al-Shaer, 2006), we can securely share these rules directly among servers. Algorithm 3 utilizes a simple subtraction operation, yielding a zero value if a packet matches a rule in the blacklist or whitelist. This protocol ensures security against passive adversaries and, through verification strategies presented in Section 4.3.1, can be made secure against active adversaries.

Algorithm 3: firewallEval$(\langle \text{Gateway}, \gamma \rangle, \langle P_i, [\phi_j]_N^{P_i} \rangle) \rightarrow (\langle \text{Gateway}, \sigma \rangle, \langle P_i, \perp \rangle)$

```
Input: Each $P_i$ has shares $[\phi_j]_N^{P_i}$ of the firewall list, for $1 \leq i \leq m$
       and $0 \leq j \leq \eta - 1$
Output: $\sigma_j = 0$: the Gateway blocks the packet
1 Gateway
    (a) $\gamma$ derived from the incoming network packet
    (b) Send $[\gamma]_N^{P_i}$ to $P_i$, for $1 \leq i \leq m$
2 For each $P_i$:
    (a) $[\sigma_j]_N^{P_i} \leftarrow [\phi_j]_N^{P_i} - [\gamma]_N^{P_i}$, for $0 \leq j \leq \eta - 1$
    (b) Send $[\sigma_j]_N^{P_i}$ to Gateway
3 Gateway
    (a) $\sigma_j \leftarrow \text{rebuild}\left([\sigma]_N^{P_1}, \dots, [\sigma]_N^{P_m}\right)$, for $0 \leq j \leq \eta - 1$
    (b) If $\exists \sigma_j = 0$: block packet
        else: forward packet
```

4.5.0.1 Handling Type B rules: The rules can be partitioned into comparison based component (CC) and non-comparison based component (NCC). There are two steps to evaluate the rules. First, we evaluate the NCC component using either firewall evaluation strategy discussed previously. After that, we will perform secure comparisons on the CC component. If either step returns false, we block the packet. Otherwise, the packet will be forwarded to the internal network.

5. Performance evaluation

We empirically analyze the computational overhead as well as the amount of data transferred between the participants for the proposed protocols. First, we clarify the storage requirement of each firewall server to store the secretly shared firewalls. As analyzed previously, the Bloom filter size is denoted by β, and each entry of the filter is secretly shared among three servers. Since the size of each share is bounded by $\log_2 N$, the storage requirement for each server is about $\beta \log_2 N$ bits. Following our experiment setup the size of the firewall rule list is $\eta = 2000$ and β is about 200K and the number of hash functions is around 20. As a result, in this specific case, we set N to be 21 and each server needs $200000 \times \log_2(21)$ bits which is about 0.108M bytes. The rest of the performance was conducted on Chameleon Cloud (Keahey et al., 2020) with three computers each with the following specifications:

- Operating system: Ubuntu Linux, Version = 18.04 LTS (Bionic Beaver).
- Machines hardware: x86 64; Cores: 12; Threads per core 2; Model: Intel®Xeon®CPU E5-2670 v3 @ 2.30 GHz, RAM 128 GB.

• The ping between machines was roughly 0.33 ms.

We evaluate the run-time for different scenarios over three servers in a distributed environment for both Replicated (or Additive) Secret Sharing (RSS) (Eerikson et al., 2020) and Shamir Secret Sharing (SSS) (Lindell and Nof, 2017) schemes under passive. We only secret share the extracted information from the packet header such as (IP address, port, and protocol type). We implemented our proposed protocols using the state-of-the-art library MP-SPDZ (Keller, 2020). It is worth mentioning that MP-SPDZ offers a parallel processing mode which we take advantage of and we set the number of threads to 8.

5.1. Complexity analysis

We provide a comprehensive analysis of the proposed protocols, encompassing both theoretical complexity and practical performance. Our approach leverages a novel MPC-based design that minimizes communication overhead. By employing secure subtraction operations performed locally and independently by each server, we significantly reduce inter-server communication, except for essential packet reception and result dissemination. This efficient design is further corroborated by our analysis of communication complexity, focusing on the number of communication rounds and data transmission required between servers presented in Lemmas 1, 2, and 3.

Lemma 1. *The firewall initialization phase presented in algorithm 1 requires sending* $(n \times \beta \times \kappa)$ *bits of communication in a partial round (only sent to the servers).*

Proof. Step 1-d sends a bloom filter of size β as a secret shares value of κ bits per secret to n servers. Given that we require $n = 3$ servers and $\kappa = 20$ bits for a bloom filter of size $\beta = 67094$, the total data to be sent from the gateway to the servers during the initialization phase adds up to 0.48 MB. □

Lemma 2. *The firewall evaluation protocol presented in algorithm 2 requires 1 round with* $m(n \times \ell + 104)$ *bits of communication.*

Proof. The evaluation phase (Step 1-b) involves transmitting a secret share of γ using 104 bits. Subsequently, Step 2-b necessitates ℓ bits to transmit the evaluation result for each packet from each server. With m packets and n servers, the total data transfer amounts to $m(n \times \ell + 104)$ bits. For instance, using $n = 3$ servers and $\ell = 20$ bits, the protocol requires 164 bits of data to be transmitted per packet. □

Lemma 3. *The firewall evaluation presented in algorithm 3 requires one round with* $n \times \ell(m + 1)$ *bits of communication.*

Proof. During the evaluation phase, Step 1-b distributes a secret share of ℓ bits per secret to n servers. Subsequently, in Step 2-b, the servers transmit their results back to the gateway, sending a total of $n \times \ell$ bits. With $n = 3$ servers and $\ell = 104$ bits per packet, the total data transferred during a single round amounts to $312(m + 1)$ bits. This represents the combined communication overhead for both the distribution and retrieval phases. □

5.2. Obliv-FW initialization

We analyze the run-time of the bloom filter initialization process. The initialization process is affected by three factors, the bloom filter size β, the number of the hash functions κ, and the size of the firewall list η.

We vary β from 50 K to 200 K. We fix the size η of the firewall list to 2000 while $\kappa = 20$. Note that in practice, η is generally in hundreds. Thus, 2000 is sufficiently large for most practical applications. Fig. 2 shows that the RSS scheme performs twice faster than the SSS scheme in all the tested values of β. On the other hand, the maximum time required for this process was by the SSS scheme which is still less than 0.08 s at the highest value of β =200K.

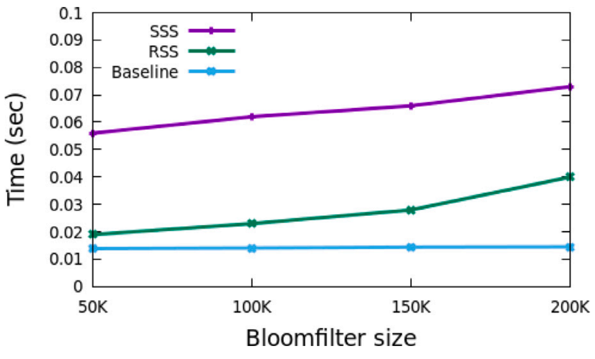

Fig. 2. Run-time varying β.

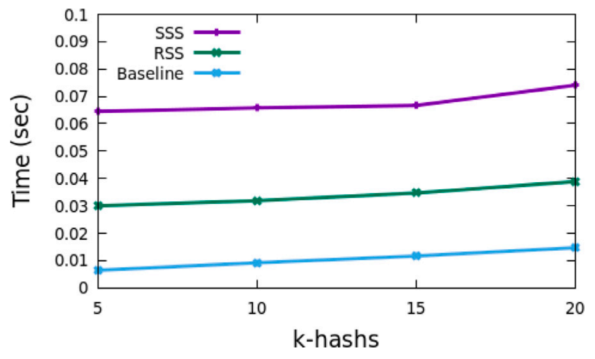

Fig. 3. Run-time varying κ.

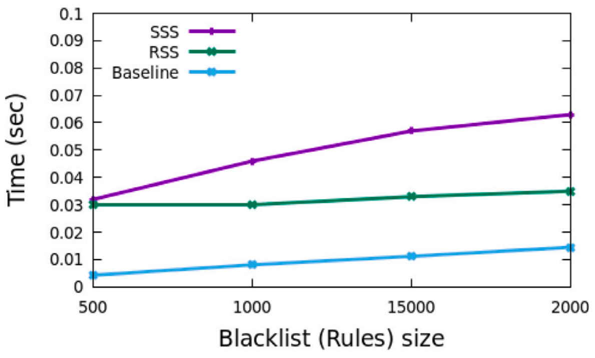

Fig. 4. Run-time varying η.

We range κ from 5 to 20 while β = 200K and η = 2000. Fig. 3 suggests that the run-time of the SSS scheme requires double the run-time of the RSS scheme for all the tested values of κ. Even though, the SSS scheme shows the highest run-time; however, it takes roughly 0.08s at the highest κ.

We vary η from 500 to 2000. While β is set to 200K and κ is fixed at 20. It is obvious from Fig. 4 that the RSS scheme has a smooth and stable performance for all the different values of η. The RSS scheme shows a linear performance of approximately 0.03s. On the other hand, the SSS scheme grows slightly with the size of η. At the highest value of η, the SSS scheme doubles the run-time compared to the RSS scheme.

We evaluate the amount of data transferred during the bloom filter initialization process. The experiment setting follows the exact settings of the previous experiments. Figs. 5, 6, and 7 show the amount of data sent during the initialization process for different values of β, κ and η

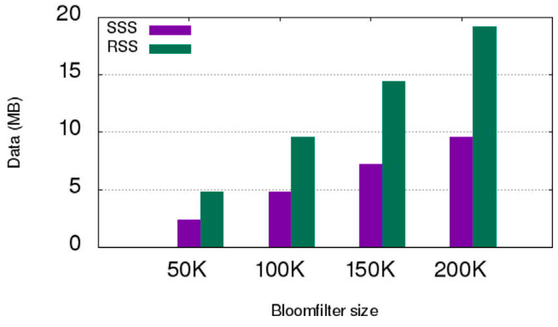

Fig. 5. Communication varying β.

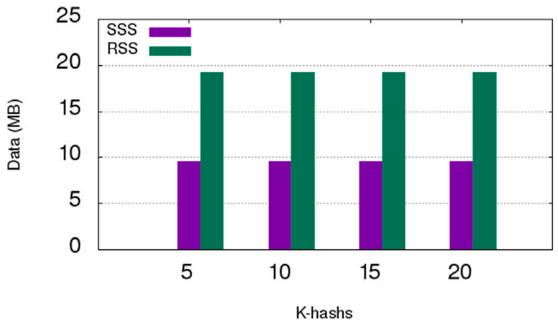

Fig. 6. Communication varying κ.

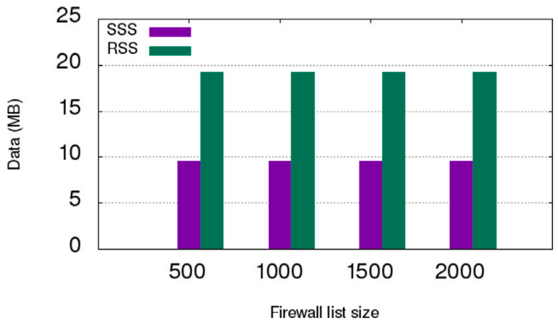

Fig. 7. Communication varying η.

respectively. It is obvious from Figs. 6 and 7 that κ and η do not affect the amount of data required to be sent during the process. The reason behind this is that the initialization involves dealing and sending shares of the bloom filter; thus, it only gets affected by β which is fixed in both cases. In all the given figures, the RSS scheme doubles the amount of data required to be exchanged by the SSS scheme. This is because RSS generates replicated additive shares and requires sending two field elements for each server, while the SSS scheme only requires sending one field element per server. It is worth mentioning that the amount of data transferred and the run-time during the initialization process are not affected by the adversary model since it only involves dealing with shares of the bloom filter and sending these shares to the correspondent servers.

Empirical evaluation across all range of scenarios demonstrates that the baseline, non-secure implementation of (Obliv-FW), exhibits a performance advantage, requiring approximately one-third of the execution time compared to the RSSS secret sharing schemes. This performance overhead, a predictable consequence of the enhanced security provided by our proposed schemes, highlights the inherent trade-off between security and efficiency.

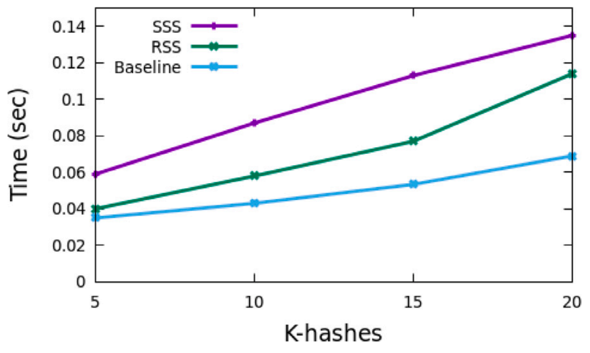

Fig. 8. Run-time varying κ, passive adversary.

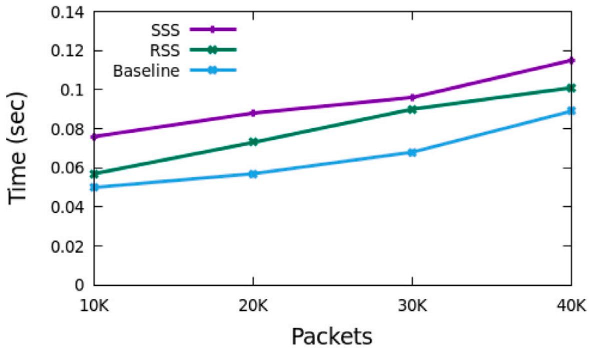

Fig. 9. Run-time varying packets, passive adversary.

5.3. *Obliv-FW evaluation*

The run-time of firewall evaluation only depends on the number of hash functions κ and the number of packets that are required to be filtered. We fix the size of the bloom filter $\beta = 200$K. We emphasize that β has no effect here.

5.3.1. *Obliv-FW evaluation (Algorithm 2)*

To show the performance of our Bloom filter-based evaluation protocol (algorithm 2) under different κ, we fix the number of the packets at 40K and vary κ from 5 to 20. On the other hand, to analyze the run-time for various numbers of packets, we set κ at 20 and range the packets from 10K to 40K. The performance of the firewall evaluation process under passive and active adversaries is summarized below.

Figs. 8 and 9 show the run-time for different κ and packets under passive adversary. In both figures, the RSS scheme is more efficient than the SSS scheme. With that being stressed, the SSS scheme requires less than 0.14 s at the highest $\kappa = 20$ for 40 K packets.

We evaluate the amount of data transferred during the evaluation process. The experiment setting follows the same settings as in the previous experiments. Figs. 10 and 11 depict the amount of data sent during the evaluation process for different values of κ and various numbers of packets consecutively under the passive adversary. Fig. 10 shows that κ does not affect the amount of data required to be sent during the evaluation process. The figures also suggest that the SSS scheme sends the same amount of data required by the RSS scheme. This is because algorithm 2 involves only local operations and the reveal step under the passive adversary is done by letting each server send its shares to the gateway which costs the same communication on both schemes.

5.3.2. *Obliv-FW evaluation (Algorithm 3)*

In this protocol, the servers simply carry a local operation that involves only shared subtraction. We fixed the size of η to 2000. We

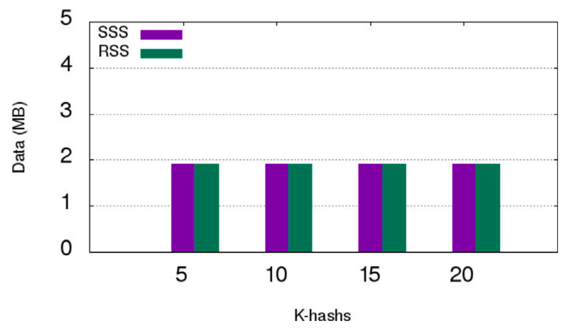

Fig. 10. Communication varying κ, passive adversary.

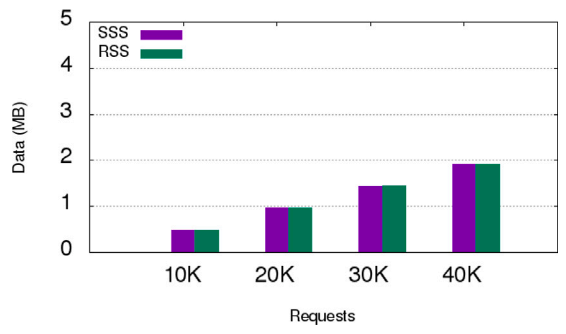

Fig. 11. Comm. varying # packets, passive adversary.

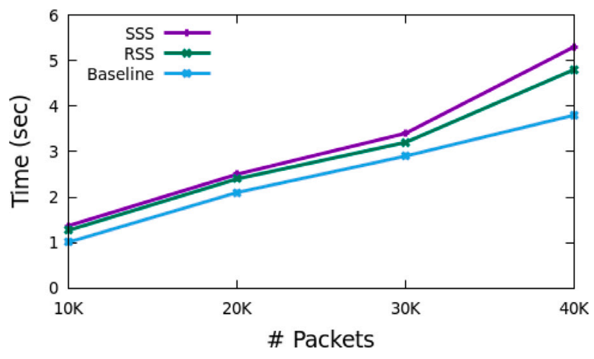

Fig. 12. Run-time varying packets, passive adversary.

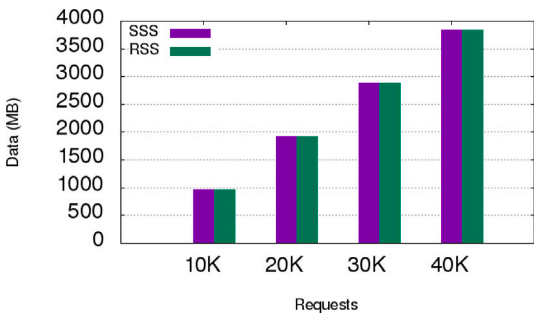

Fig. 13. Comm. varying packets, passive adversary.

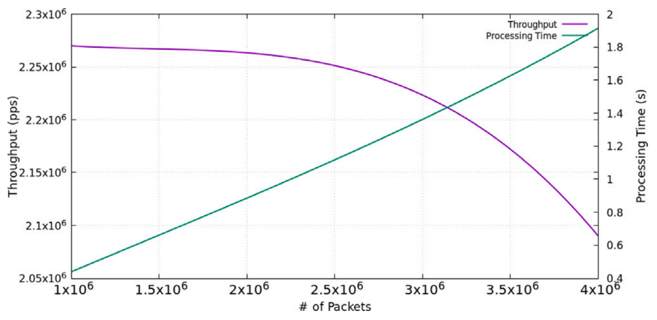

Fig. 14. Throughput as # of Packets increases.

vary the number of packets from 10 K to 40 K following the previous experiment setting. Fig. 12 depicts the run-time for different numbers of packets under the passive adversary setting. The figure illustrates the same performance for both RSS and SSS schemes. At the maximum number of packets, RSS and SSS schemes require roughly 5 s to evaluate 40K packets. 5 s seem to be a large overhead. However, it is only 0.000125s per packet. In addition, since the firewall rules do not change frequently, known packets could be buffered in the memory, and there is no need to perform a secure firewall evaluation for these packets if the firewall rules are not updated. As a result, the overhead of the proposed protocol can be negligible.

We evaluate the amount of data transferred during the evaluation process of algorithm 3. We set the experiment the same way as in the previous tests. Fig. 13 demonstrates the amount of data sent during the evaluation process for different numbers of packets under the passive adversary setting. The figure shows that both schemes require to send the same amount of data. The performance data for active adversaries show similar trends as in the previous experiments. We omit those figures due to limited space.

The baseline non-secure implementation (Obliv-FW) outperforms the RSSS secret sharing scheme in terms of execution time, achieving approximately half the execution time. This performance advantage, however, is directly correlated with the enhanced security provided by our proposed schemes.

5.4. *Throughput and bandwidth*

In our proposed solution, the throughput of the firewall system, measured in packets per second, is influenced by both the number of packets being processed and the number of hash functions, denoted by κ. Fig. 14 shows an analysis of the relationship between throughput and processing time as the packet volume increases which reveals a linear growth in processing time. This suggests a consistent processing overhead per packet, implying minimal queuing delays or non-linear computational complexity.

The data shows a minimal decrease in throughput as the number of packets increases, from 2.27 million packets per second (pps) at 1 million packets to 2.09 million pps at 4 million packets. This relatively small change in throughput suggests that the system is capable of handling a significant increase in packet volume with only a slight performance degradation, indicating good scalability and efficiency. The minimal decrease in throughput suggests that the system is well-optimized to handle a growing workload, due to the efficient protocol that offers effective resource utilization.

Fig. 15 indicates a gradual decrease in throughput as the number of hash functions (k) increases. While the throughput drops from 6.29 million packets per second (pps) with k=5 to 2.19 million pps with k=20, the processing time remains relatively consistent, suggesting that the system's computational cost is not significantly impacted by the increased complexity. Despite this decrease, the system still demonstrates a reasonable level of performance with a higher number of hash functions. Further analysis could explore the factors contributing to the slight throughput reduction, such as network congestion or computational overhead, to pinpoint potential areas for optimization and further improve the system's scalability.

We examine the bandwidth requirements for the proposed Obliv-FW due to the utilization of three servers with secret sharing for the upcoming packets. Fig. 16 demonstrates a positive trend in our

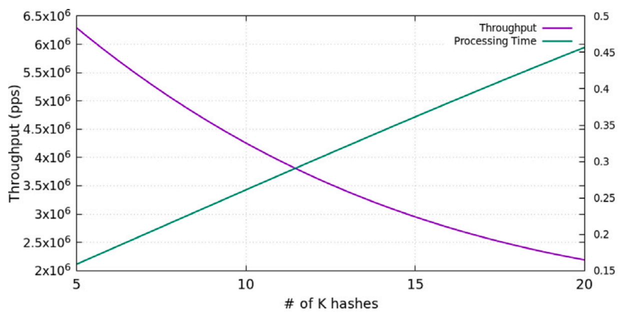

Fig. 15. Throughput as # of κ increases.

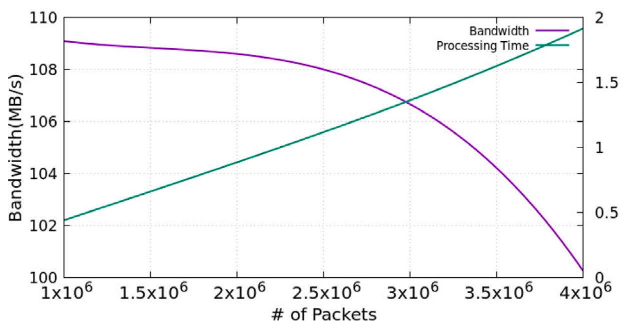

Fig. 16. Bandwidth as # of Packets increases.

Table 3
The effect of FP rate on β and κ.

FP	β	κ	Memory(MB)
0.001	28 754	10	0.0377
0.0001	38 339	14	0.0686
0.00001	47 924	17	0.1030
0.000001	57 509	20	0.1441
0.0000001	67 094	24	0.2004

system's bandwidth efficiency, showcasing a decrease in bandwidth required despite an increase in the number of packets processed. While processing time increases linearly with packet volume, the bandwidth remains relatively stable, suggesting that the system effectively utilizes available bandwidth resources even with a growing workload. This efficient bandwidth utilization is a positive sign of our system's design, highlighting its ability to handle larger traffic volumes without significantly increasing bandwidth consumption.

5.5. *Effects of false positive (FP) rate*

Table 3 shows different values of β, κ, and memory (MB) under various FP rates given $\eta = 2000$. According to the empirical results, the FP rate does not affect the run-time very much. For example, to lower the FP rate from 0.001 to 0.000001, the run-time is only doubled as shown in Figs. 8. Despite an increase in memory requirement with decreasing false positive (FP) rates, the presented data reveals that even at an exceptionally low FP rate of $1e^{-7}$, the memory requirement remains relatively modest, at approximately 0.201 MB. This highlights the inherent efficiency of Bloom filters, enabling the achievement of low FP rates with a small memory footprint.

6. Discussion

While Obliv-FW presents a novel approach to securing firewall systems with strong guarantees, several limitations and avenues for future research exist.

First, our assumption requires administrators to be trusted. While relying on trusted administrators aligns with many established frameworks like Embark, it also presents certain level of vulnerability. To reduce dependence on the trustworthiness of individual administrators, one can add other security layers, such as multi-factor authentication and maintaining detailed audit logs of all administrator actions.

Second, Obliv-FW mitigates insider threats by distributing trust and secret shares among multiple servers. In the extreme case where the number of compromised servers exceeds our threshold t, we will need to integrate other security measures such as outlier detection.

Third, our current evaluation focuses on protocol processing time. When deploying Obliv-FW in the real world settings, fluctuating network conditions and congestion may impact the latency and performance of Obliv-FW, which will require a long term testing.

7. Conclusion

Our contribution in this paper amounts to developing protocols for a secure, distributed, and oblivious firewall evaluation. We have achieved this new and secure firewall management paradigm through the use of secret sharing schemes as well as Bloom filters to represent the firewall. We also proposed multiple means for implementing this functionality with various options to increase efficiency or security, including fault tolerance and security against malicious attempts to disrupt the system. Furthermore, we have shown that our proposed approach needs not to be static but can be dynamically updated if necessary to handle new threats or other security issues. Our experimental results have shown that the overhead associated with the protocols is reasonable depending on design choices and security concerns. Securing firewalls against external threats is only the first step, and we are planning to adopt the proposed methodology to develop a suite of secure and distributed protocols for managing network functionalities and resources, such as access control, user management, and resource allocations.

CRediT authorship contribution statement

Ali Allami: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Data curation, Conceptualization. **Tyler Nicewarner:** Writing – review & editing, Visualization, Validation, Data curation. **Ken Goss:** Writing – review & editing, Writing – original draft, Visualization, Validation, Formal analysis, Data curation, Conceptualization. **Ashish Kundu:** Validation, Investigation, Conceptualization. **Wei Jiang:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Formal analysis, Conceptualization. **Dan Lin:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. More results

A.0.1. Oblivious firewall evaluation for Type B rules

In this protocol, the servers additionally perform two secure comparisons (Damgård et al., 2006) one for the upper bound and the other for the lower bound. We fixed the size of η to 2000. We vary the number of packets from 1 to 1k. Fig. 17 illustrates the run-time for various packets under the passive adversary setup. The figure shows that RSS is roughly 21 times faster than the SSS scheme to evaluate 1K packets. The time seems to be a large overhead. However, it is only 0.012s per packet. As we have mentioned previously the known packets can be buffered in the memory, and there is no need to perform secure firewall evaluation for these packets if the firewall rules are not updated. Therefore, the overhead in this case can be negligible.

Fig. 18 depicts the amount of data sent during the evaluation process for different numbers of packets under the passive adversary setting. The figure shows that the RSS scheme requires sending 17 times less data than the SSS scheme.

Appendix B. Secret sharing

Secret Sharing schemes enable distributed computation of data by dividing it into shares. This technique has applications in secure multi-party computation (MPC), such as solving Yao's Millionaire problem (Yao, 1982, 1986). Inspired by this foundational concept, we propose a solution to address the challenges outlined earlier.

B.1. Shamir's secret sharing scheme

This work focuses on Shamir's secret sharing scheme [Shamir, 1979]. In this scheme, data is encoded as the constant term of a degree-t polynomial, where t is a threshold value. For minimal complexity and no need for multiplications, m (number of servers) can be set to $t+1$. For scenarios involving multiplication, m should be set to a value greater than $2t$.

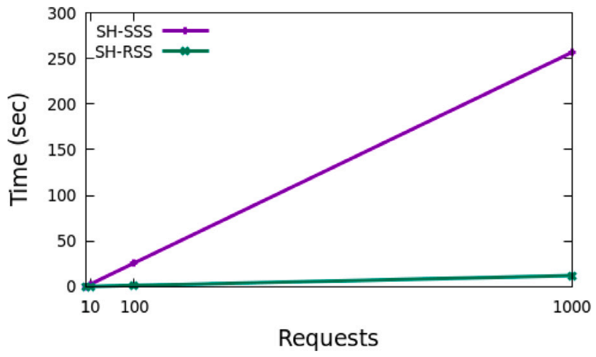

Fig. 17. Run-time varying packets, passive adversary.

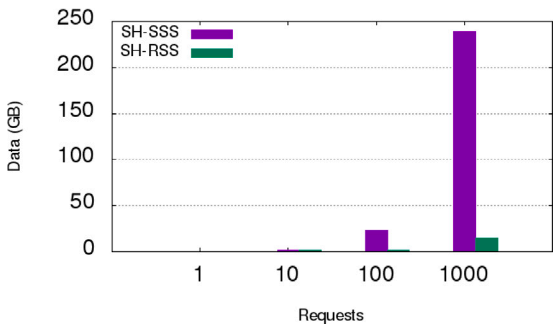

Fig. 18. Comm. varying packets, passive adversary.

B.1.0.1. Generating shares Formally, a secret value s is placed within a t degree polynomial f of the form: $f(x) = c_t x^t + c_{t-1} x^{t-1} + \cdots + c_2 x^2 + c_1 x + s \mod N$ in which all constants c_t through c_1 are chosen from a uniform random distribution on the domain denoted by $\mathbb{Z}_N$ where N is a prime and sufficiently large to represent s. For all m servers, each server P_i's share of s is constructed by: $[s]_N^{P_i} = f(i) \mod N$

B.1.0.2. Rebuilding secrets Lagrange Polynomial Interpolation is used to reconstruct f and subsequently derive s: $s = f(0) = \sum_{j=1}^{m} [s]_N^{P_j} \prod_{k=1:k\neq j}^{m} \frac{-x_k}{x_j - x_k} \mod N$ Any $t+1$ valid shares can be used to reconstruct s.

B.1.0.3. Addition In the Shamir Secret Sharing Scheme, addition is free in that no communication is required between servers, and no lengthy local computations are necessary. For two values, a and b, which are shared among the servers, the shared sum is computed by each server P_i locally computing their new share by adding their shares of the secrets a and b: $[a+b]_N^{P_i} = [a]_N^{P_i} + [b]_N^{P_i} \mod N$. The addition of a public constant c is affected simply through the addition of this constant into each party's shares $[a+c]_N^{P_i} = [a]_N^{P_i} + c$

Data availability

No data was used for the research described in the article.

References

Al-Shaer, Ehab S., Hamed, Hazem H., 2004. Modeling and management of firewall policies. IEEE Trans. Netw. Serv. Manage. 1 (1), 2–10.

Aumasson, Jean-Philippe, Bernstein, Daniel J., 2012. SipHash: a fast short-input PRF. In: International Conference on Cryptology in India. Springer, pp. 489–508.

Broder, Andrei, Mitzenmacher, Michael, 2004. Network applications of bloom filters: A survey. Internet Math. 1 (4), 485–509.

Cohn, Henry, Heninger, Nadia, 2013. Approximate common divisors via lattices. In: The Open Book Series, vol. 1, (1), Mathematical Sciences Publishers, pp. 271–293.

Curtmola, Reza, Garay, Juan, Kamara, Seny, Ostrovsky, Rafail, 2006. Searchable symmetric encryption: Improved definitions and efficient constructions. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. ACM, pp. 79–88.

Damgård, Ivan, Fitzi, Matthias, Kiltz, Eike, Nielsen, Jesper Buus, Toft, Tomas, 2006. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Theory of Cryptography Conference. Springer, pp. 285–304.

Duan, Huayi, Wang, Cong, Yuan, Xingliang, Zhou, Yifan, Wang, Qian, Ren, Kui, 2019. LightBox: Full-stack protected stateful middlebox at lightning speed. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 2351–2367.

Eerikson, Hendrik, Keller, Marcel, Orlandi, Claudio, Pullonen, Pille, Puura, Joonas, Simkin, Mark, 2020. Use your brain! arithmetic 3pc for any modulus with active security. In: 1st Conference on Information-Theoretic Cryptography. ITC 2020, Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Einziger, Gil, Friedman, Roy, 2017. TinySet—An access efficient self adjusting bloom filter construction. IEEE/ACM Trans. Netw. 25 (4), 2295–2307.

Filipek, Jozef, Hudec, Ladislav, 2016. Securing mobile ad hoc networks using distributed firewall with PKI. In: Applied Machine Intelligence and Informatics (SAMI), 2016 IEEE 14th International Symposium on. IEEE, pp. 321–325.

GDPR.eu, 2023. What is GDPR?. https://gdpr.eu/what-is-gdpr/. (Accessed: 27 October 2023).

Geravand, Shahabeddin, Ahmadi, Mahmood, 2013. Bloom filter applications in network security: A state-of-the-art survey. Comput. Netw. 57 (18), 4047–4064.

Goldreich, Oded, 2004. Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press.

Google, 2017. IPv6 – Google. https://www.google.com/intl/en/ipv6/statistics.html. [Online; Accessed 26 April 2018].

Gouda, Mohamed G., Liu, X.-Y.A., 2004. Firewall design: Consistency, completeness, and compactness. In: Distributed Computing Systems, 2004. Proceedings. 24th International Conference on. IEEE, pp. 320–327.

Grubbs, Paul, Arun, Arasu, Zhang, Ye, Bonneau, Joseph, Walfish, Michael, 2022. {Zero-knowledge} middleboxes. In: 31st USENIX Security Symposium. USENIX Security 22, pp. 4255–4272.

Guruswami, Venkatesan, Sudan, Madhu, 1998. Improved decoding of reed-solomon and algebraic-geometric codes. In: Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280). IEEE, pp. 28–37.
Hamed, Hazem, Al-Shaer, Ehab, 2006. Dynamic rule-ordering optimization for high-speed firewall filtering. In: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security. pp. 332–342.
Han, Juhyeng, Kim, Seongmin, Cho, Daeyang, Choi, Byungkwon, Ha, Jaehyeong, Han, Dongsu, 2020. A secure middlebox framework for enabling visibility over multiple encryption protocols. IEEE/ACM Trans. Netw. 28 (6), 2727–2740.
Internet Society, 2018. State of IPv6 Deployment 2017 | Internet Society. Internet Soc. https://www.internetsociety.org/resources/doc/2017/state-of-ipv6-deployment-2017. [Online; Accessed 26 April 2018].
Ioannidis, Sotiris, Keromytis, Angelos D., Bellovin, Steve M., Smith, Jonathan M., 2000. Implementing a distributed firewall. In: Proceedings of the 7th ACM Conference on Computer and Communications Security. ACM, pp. 190–199.
Kaur, Karamjeet, Kumar, Krishan, Singh, Japinder, Ghumman, Navtej Singh, 2015. Programmable firewall using software defined networking. In: Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on. IEEE, pp. 2125–2129.
Keahey, Kate, Anderson, Jason, Zhen, Zhuo, Riteau, Pierre, Ruth, Paul, Stanzione, Dan, Cevik, Mert, Colleran, Jacob, Gunawi, Haryadi S., Hammock, Cody, Mambretti, Joe, Barnes, Alexander, Halbach, François, Rocha, Alex, Stubbs, Joe, 2020. Lessons learned from the chameleon testbed. In: Proceedings of the 2020 USENIX Annual Technical Conference. USENIX ATC '20, USENIX Association.
Keller, Marcel, 2020. MP-SPDZ: A versatile framework for multi-party computation. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security.
Khakpour, Alireza Rahmati, Liu, Albert X., 2012. First step toward cloud-based firewalling. In: 2012 IEEE 31st International Symposium on Reliable Distributed Systems. IEEE, pp. 41–50.
Kreutz, Diego, Ramos, Fernando MV, Verissimo, Paulo Esteves, Rothenberg, Christian Esteve, Azodolmolky, Siamak, Uhlig, Steve, 2015. Software-defined networking: A comprehensive survey. Proc. IEEE 103 (1), 14–76.
Kurose, James F., Ross, Keith W., 2017. Computer Networking: A Top-Down Approach, seventh ed. Pearson.
Lan, Chang, Sherry, Justine, Popa, Raluca Ada, Ratnasamy, Sylvia, Liu, Zhi, 2016. Embark: Securely outsourcing middleboxes to the cloud. In: 13th USENIX Symposium on Networked Systems Design and Implementation. NSDI 16, pp. 255–273.
Lindell, Yehuda, Nof, Ariel, 2017. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 259–276.
Maccari, Leonardo, Fantacci, Romano, Neira, P., Gasca, Rafael Martinez, 2007. Mesh network firewalling with bloom filters. In: Communications, 2007. ICC'07. IEEE International Conference on. IEEE, pp. 1546–1551.
Neira, P., Gasca, Rafael Martinez, Maccari, Leonardo, Lefèvre, Laurent, 2008. Stateful firewalling for wireless mesh networks. In: New Technologies, Mobility and Security, 2008. NTMS'08, IEEE, pp. 1–5.
NIST, 2001. Technical Report NIST Special Publication FIPS-197, National Institute of Standards and Technology, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
Peltier, Thomas R., Peltier, Justin, 2006. Complete Guide to CISM Certification. Auerbach Publications.
Pena, Justin Gregory V., Yu, William Emmanuel, 2014. Development of a distributed firewall using software defined networking technology. In: Information Science and Technology (ICIST), 2014 4th IEEE International Conference on. IEEE, pp. 449–452.
Poddar, Rishabh, Lan, Chang, Popa, Raluca Ada, Ratnasamy, Sylvia, 2018. {SafeBricks}: shielding network functions in the cloud. In: 15th USENIX Symposium on Networked Systems Design and Implementation. NSDI 18, pp. 201–216.
Satasiya, Dhaval, Raviya, Rupal, Kumar, Hiresh, 2016. Enhanced SDN security using firewall in a distributed scenario. In: Advanced Communication Control and Computing Technologies (ICACCCT), 2016 International Conference on. IEEE, pp. 588–592.
Shamir, Adi, 1979. How to share a secret. Commun. ACM 22 (11), 612–613.
Sherry, Justine, Lan, Chang, Popa, Raluca Ada, Ratnasamy, Sylvia, 2015. BlindBox: Deep packet inspection over encrypted traffic. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. ACM, pp. 213–226.
Tarkoma, Sasu, Rothenberg, Christian Esteve, Lagerspetz, Eemil, 2012. Theory and practice of bloom filters for distributed systems. IEEE Commun. Surv. Tutor. 14 (1), 131–155.
Trach, Bohdan, Krohmer, Alfred, Gregor, Franz, Arnautov, Sergei, Bhatotia, Pramod, Fetzer, Christof, 2018. ShieldBox: Secure middleboxes using shielded execution. In: Symposium on SDN Research. ACM, p. 2.
Welch, Lloyd R., Berlekamp, Elwyn R., 1986. Error correction for algebraic block codes. Google Patents, US Patent 4, 633, 470.
Yao, Andrew Chi-Chih, 1982. Protocols for secure computations. In: Foundations of Computer Science, vol. 82, IEEE, pp. 160–164.
Yao, Andrew Chi-Chih, 1986. How to generate and exchange secrets. In: Foundations of Computer Science. IEEE, pp. 162–167.
Zhang, Collin, DeStefano, Zachary, Arun, Arasu, Bonneau, Joseph, Grubbs, Paul, Walfish, Michael, 2023. Zombie: Middleboxes that don't snoop. Cryptology ePrint Archive.

Ali Allami is currently a postdoctoral research scholar in the Department of Computer Science at Vanderbilt University. He received a Ph.D. degree in computer science from the University of Missouri-Columbia in 2021. His research interests include Secure Multiparty Computation and Privacy-Preserving data analysis.

Tyler Nicewarner received his B.S. degree in 2020 from the University of Missouri-Columbia. He is currently working toward a Ph.D. degree in Computer Science at Vanderbilt University. His research interests include Secure Multiparty Computation, Privacy Preservation, and Machine Learning.

Ken Goss is currently a Senior Research and Development Scientist at the Sandia National Laboratories. He received a Ph.D. degree in computer science from the University of Missouri-Columbia in 2020. His research interests include Security, Privacy, and Machine Learning.

Ashish Kundu is currently the Head of Cybersecurity Research at Cisco Research. He received his Ph.D. degree in Computer Science from Purdue University. His research interests cover many areas in the fields of Security, Privacy, and AI.

Wei Jiang is currently a research scientist at Oracle Labs. He received his BS in computer science from the University of Iowa and a Ph.D. in Computer science from Purdue University. His research interests include privacy-preserving data analysis and applied cryptography.

Dan Lin is currently a professor in the Department of Computer Science and Director of the I-Privacy Lab at Vanderbilt University. She received her Ph.D. degree in Computer Science from the National University of Singapore in 2007 and was a postdoctoral research associate at Purdue University for two years. Her research interests cover many areas in the fields of information security and database systems.