Enabling Privacy-Preserving Header Matching for Outsourced Middleboxes

Yu Guo*, Cong Wang*, Xingliang Yuan†, and Xiaohua Jia*
*City University of Hong Kong, Hong Kong, China
†Monash University, Melbourne, Australia
y.guo@my.cityu.edu.hk, {congwang, csjia}@cityu.edu.hk, xingliang.yuan@monash.edu

***Abstract*—Over the past few years, enterprises start adopting software middlebox services from cloud or NFV service providers. Although this new service model is recognized to be cost-effective and scalable for traffic processing, privacy concerns arise because of traffic redirection to outsourced middleboxes. To ease these concerns, recent efforts are made to design secure middlebox services that can directly function over encrypted traffic and middlebox rules. But prior designs only work for portions of frequently-used network functions. To push forward this area, in this work, we investigate header matching based functions like firewall filtering and packet classification. To enable privacy-preserving processing on encrypted packets, we start from the latest primitive "order-revealing encryption (ORE)" for encrypted range search. In particular, we devise a new practical ORE construction tailored for network functions. The advantages include: 1) guaranteed protection of packet headers and rule specified ranges; 2) reduced accessible information during comparisons; 3) rule-aware size reduction for ORE ciphertexts. We implement a fully functional system prototype and deploy it at Microsoft Azure Cloud. Evaluation results show that our system can achieve per packet matching latency 0.53 to 15.87 millisecond over 1.6K firewall rules.**

I. Introduction

Network middleboxes like application firewalls, NATs and intrusion detection [1], [2] have found widespread adoption in enterprise networks. Due to the fast-evolving network infrastructures and functions, managing in-house hardware middleboxes have been regarded neither economical nor scalable. Therefore, software (virtualized) middlebox services [2] are rapidly developed with immediate benefits of reduced capital cost, ease of management, scaled deployment, etc.

However, this trending paradigm raises security and privacy concerns because of its intrinsic service model. That is, enterprise gateways have to redirect traffic to middlebox service providers for processing and actions [2]. The consequence is that traffic contents like packet headers and payloads are unwillingly revealed. Exploiting packet payloads may extract private information of enterprises, e.g., business secrets [3], and analyzing packet headers may derive critical information of enterprise infrastructures, e.g., network topology [4]. In addition, middlebox rules are usually customized by enterprises or third-party specialists [3], [5]. They are private or proprietary which should be protected.

We note that standard end-to-end encrypted communication protocols like HTTPs and IPsec guarantee the confidentiality of traffic contents, but prevent middleboxes from processing encrypted traffic. A plausible method is to intercept and decrypt the traffic in middleboxes [2]. Unfortunately, it violates confidentiality guarantees, because traffic is still processed over middlebox rules in cleartext.

To address the above issues, recent efforts [3]–[5] are made to design middlebox systems that can directly function over encrypted traffic and rules. Sherry *et al.* [3] and Yuan *et al.* [5] propose protocols respectively for encrypted pattern matching to support functions like deep packet inspection, but they do not consider functions based on range matching like header classification and firewall filtering. To fill the gap, Lan *et al.* devise a prefix encryption scheme to enable prefix rule matching over encrypted headers. But their solution also has limitations in functionality. For example, header classification does not always rely on IP addresses being prefixed, e.g., non-contiguous masks like 137.98.217.0/8.22.160.80 [6]. Another example is that inspection in some fields of headers usually requires a wide variety of range specifications, e.g., a port number greater than 1021, or range 16-20 [1]. Representing these ranges in prefixes will cause a significant increase in the number of rules, rending inspection throughput downgrade.

Motivated by the observations above, our goal in this work is to design a privacy-preserving yet generic scheme for encrypted packet header matching in outsourced middlebox services. To preserve functionality, encryption schemes that allow range matching on ciphertext are desirable. One possible candidate is order-preserving encryption (OPE) [7], where OPE ciphertext inherits the order relations of the underlying values so that comparisons can directly be applied. But OPE leaks partial information of plaintext values and distances between them [7], and is vulnerable to frequency analysis [8].

To improve security, the notion of order-revealing encryption (ORE) [9], [10] is proposed. ORE ciphertext achieves semantic security, while order relations are learned during dedicated comparison protocols. By applying ORE schemes to our context, ranges defined in middlebox rules are protected against the attackers who can access ORE ciphertexts only. Namely, an outside attacker who steals and dumps encrypted middlebox rules will not derive any useful information. But we note that prior ORE comparison protocols indicate partial information of the underlying values, aka the most significant bits/blocks that differ between two ciphertexts. Directly using current ORE schemes appears to suffer from the latest leakage-abuse attacks [11], [12], which compromises the privacy of

978-1-5386-2542-2/18/$31.00

packet headers and middlebox rules.

Therefore, a scheme with reduced leakage should be designed to serve for encrypted range matching. To achieve this goal, our observation lies in the following two aspects. First, it is possible to mitigate the aforementioned leakage, aka the most significant differing-bits/blocks of the plaintexts, as demonstrated by Cash *et al.* in [13]. Second, order relations are not necessarily revealed for range matching, which can also be ciphertexts, as shown by Yuan *et al.* in [14]. Unfortunately, the scheme proposed in [13] involves cryptographic pairing operations, which can hardly satisfy the performance requirements in network functions. Yuan *et al.*'s scheme is an enhanced version of the Lewi and Wu's scheme [10] which also has the above leakage.

By leveraging the design philosophy of the two studies, we devise a new ORE scheme that achieves both performance and security goals simultaneously. For efficiency, our design starts from Yuan *et al.*'s scheme with only symmetric-key based operations. From a high-level point of view, values are encrypted into ciphertext blocks, where each block also embeds the prefix of the block and the tokens of the order relations. To protect the location of the most significant block that differs, we follow the methodology of Cash *et al.*'s scheme to permute those ciphertext blocks randomly. However, block permutation does not fully hide that information, because block equality revealed in comparison indicates the numbers of equal blocks across multiple comparisons, which could further be exploitable to infer order relations of two ciphertexts without comparison (see detailed analysis in Sec. IV-A). To this end, we remove the equal relations embedded in ciphertext blocks, while still preserving the correctness.

For practical considerations, we propose a rule-aware size reduction technique to shrink the size of the ORE ciphertexts. Our intuition is that header matching rules for firewall filters and traffic classification perform actions based on pre-defined ranges with a fixed order relation. Therefore, only one order relation is required for encrypted header matching. The rest of order relations do not have to be embedded. Moreover, we provide a novel encrypted data structure that aims for secure and efficient ciphertext compression while preserving strong security guarantees. The proposed design achieves both security and efficiency. A formal security analysis is given to demonstrate the security strength of this design.

In summary, our contributions are listed as follows:

- We propose an ORE scheme that enables secure and efficient range matching based network functions in outsourced middleboxes. It hides order relations and partial information leaked by prior ORE schemes.
- We devise a rule-aware size reduction technique to reduce the size of the encrypted middlebox rules. Meanwhile, we provide a novel encrypted data structure to further improve the security and space efficiency.
- We implement our system prototype and deploy it on Microsoft Azure. The evaluations on real-world rulesets show that the processing latency per packet ranges from 0.53 to 15.87 millisecond and the processing throughput reaches over 800 packets per second over 1600 rules.

The paper is organized as follows. Section II introduces related work. Section III presents our system architecture and threat model. Section IV presents our system design. Our security analysis is conducted in Section V, and an extensive array of evaluation results is shown in Section VI. Finally, we conclude the paper in Section VII.

II. Related Work

Secure Middlebox Outsourcing: To enable secure network functions in middlebox services, a line of work is proposed in the past few years [3]–[5], [15]–[21]. Early studies [15], [16] focus on rule anonymization, and do not consider to protect traffic privacy. Sherry *et al.* [3] propose the first system for encrypted deep packet inspection. Then Yuan *et al.* [5] introduce a design for pattern-matching based functions with more dedicated rule protection. But the above two designs do not support secure range matching based functions. Recently, a system designed by Lan *et al.* [4] introduces a prefix encryption scheme to support functions via encrypted prefix matching. As mentioned, it still has limits in functionality for other range matching based functions. We note that the designs proposed by Melis *et al.* [17] and Asghar *et al.* [18] transform the computation of header matching into bitwise operations. In particular, the design in [17] proposes to leverage homomorphic encryption for secure computation, while the one in [18] proposes to leverage secure multi-party computation. Those sophisticated primitives do not appear to satisfy the performance requirements in networked contexts.

Order-revealing Encryption: Order-revealing encryption (ORE) [9], [10], [13] (just to list of few) is a cryptographic primitive to enable secure range search. They are designed to improve the security of order-preserving encryption. The ORE ciphertexts themselves do not show order relations. After the query, the order relations are learned. The first practical ORE scheme is proposed by Chenette *et al.* in [9], but it leaks the first different bit of two messages. To address this issue, Lewi *et al.* [10] propose an ORE scheme that achieves semantic security while leaks the first different bit block of two messages during comparisons. Concurrently, Cash *et al.* [13] provide another different ORE scheme based on bilinear pairing to further reduce the leakage above. That is, the location of the first block that differs is protected.

III. Overview

A. System Architecture

Figure 1 shows our system architecture, containing three entities, the client in the private enterprise network, the host in the application servers and the middlebox (*MB*) running at the cloud or NFV service providers. Our design serves the client who wishes to outsource network processing to the cloud. The client runs a gateway (*GW*) which sends encrypted traffic and tokens generated from packet headers to the outsourced middlebox. The middlebox processes encrypted traffic via

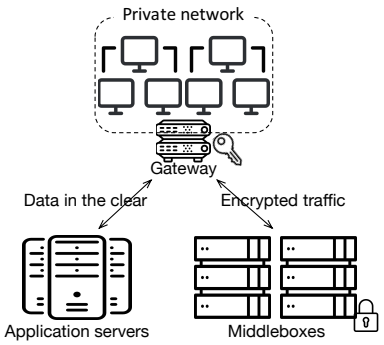


Fig. 1: System architecture

ORE comparisons and then returns encrypted results to the client. Our system functions are described as follows:

Initialization: To begin with, the gateway has a firewall ruleset to be outsourced to the middlebox. The gateway first parses endpoints from the ruleset and encrypts a rule filter based on our ORE scheme. Then it deploys the encrypted filter to the middlebox. After that, the outsourced middlebox can perform encrypted header matching over the encrypted rule.

Pre-processing: When the gateway receives a packet from the enterprise network, it will parse the packet header into a set of bit strings. Then it generates ORE tokens based on the extracted bit string. After pre-processing is completed, the packet will be redirected to the middlebox in the encrypted form for traffic processing.

Inspection: Once the encrypted packet arrives, the middlebox will hold up the packet and execute the header matching inspection between tokens and encrypted header matching rules. Once a matched rule is found, the corresponding actions, e.g., dropping packets will be executed.

B. Threat Model

We consider that the enterprise gateway is trusted. The rule encryption keys are securely stored at the gateway. We assume that threats come from semi-honest adversaries on service providers. It faithfully offers middlebox services but may intend to learn the sensitive information from the traffic and tries to exploit the proprietary middlebox rules. It can monitor the encrypted traffic and tokens. We do not consider the case that attackers can access the gateway's private keys, but they can dump all the data from outsourced middleboxes. Finally, we do not consider the case where malicious attackers modify or delete packets.

IV. THE PROPOSED SYSTEM

This section presents the design of encrypted header matching in detail, based on our enhanced order-revealing encryption scheme. Meanwhile, we show that our design works compatibly with existing range-based firewall filtering, and further introduce a secure rule-aware optimization for security and practical considerations.

A. Design Rationale

Recent work that seeks to protect the middlebox rules while preserving the range-based inspection falls into two categories. In the first category, the secure outsourced system specifically designed for some rule scenarios has been implemented, such as embark's design [4] for prefix matching inspection. But it cannot serve for all the range match based network functions. In the second category, cryptographic primitives have been designed to enable the cloud provider to perform range-based queries. Among others, one promising technique in the current literature is order-revealing encryption. The initial result, also known as order-preserving encryption, allows ciphertext to preserve the original order of the plaintext to conduct the numeric comparison. Unfortunately, orders are directly exposed in ciphertexts, which make OPE schemes are vulnerable to statistical inference attacks.

To improve the security strength, we start from a very recent work that achieves semantic secure notion for practical ORE [10], and it is also the most efficient ORE scheme currently. To get a better understanding of our idea, we first briefly recall the adopted ORE scheme [10]. Given a plaintext x, it first splits x into b blocks with equal bit length. Then for each bit block $x_{|i}$, it masks the order information $cmp(x_{|i}, y)$ via computing $X_i = cmp(x_{|i}, y) + Q(K_i, r)$, where the encryption key K_i is the pseudorandom function for prefix block $x_{|i-1}$, Q is a pseudorandom function, y is all possible value in each block and r is a nonce. The resulting ciphertext is $(r, X_1, \cdots, X_b)$. To compare a value y with the above ORE ciphertext, one needs to sequentially generate the query token $(t_1, \cdots, t_b)$, where each block token t_i is the pseudorandom value for its prefix $y_{|i-1}$. Now given tokens $(t_1, \cdots, t_b)$ with ciphertext $(r, X_1, \cdots, X_b)$, one can obtain the order from the least block i via computing $X_i - Q(t_i, r)$. For more details, we refer the readers to [10].

This treatment can support both prefix match and range search on encrypted data. But it reveals the location of the first different block as well as order/equal information. Above leakage tells partial order between ciphertexts, which can also reveal more information [13]. For instance, suppose we compare an encrypted value $x = 0000$ with two ciphertexts $y_1 = 0001$ and $y_2 = 1100$ respectively, where the block size is 2. The comparison result of $cmp(x, y_1)$ is " $<$ " and the first different block is 2 (i.e., they have the same prefix). Yet, $cmp(x, y_2)$ is " $<$ ", and the first different block is 1. Since comparisons are transitive, an attacker can infer that the value y_1 is smaller than y_2 without decrypting because of the order information and distinguishability of the first different block.

To limit above leakage, a promising solution is proposed by Cash *et al.* [13]. They propose to use bilinear map over permuted bit blocks to hide the location of the first differ block. Unfortunately, their construction requires heavy public key operations, i.e. pairing operations, which makes its performance unclear. What is more, the order information is allowed to be learned after queries. Recently, there have been several attacks using the auxiliary information such as order

Algorithm 1 $\mathsf{Build}_{\mathsf{rule}}$: Build header matching rule

Input: Private key k_1; secure PRFs $\{F, Q\}$; secure PRP $\{\pi, \phi\}$; hash function H; endpoint $v \in \{v^1, v^2\}$; all possible block value v^*; rule condition cmp.

Output: Encrypted header matching rule $Rule_v$.

1: Generate a nonce γ;
2: **for** $i \in \{1, b\}$ **do**
3: **for** $j \in \{1, 2^d\}$ **do**
4: **if** $cmp(v^*_{|i}, v_{|i}) \neq$ "$equal$" **then**
5: $s_{i,j} \leftarrow F_{k1}(v^*_{|i}||cmp(v^*_{|i}, v_{|i}), H(v_{|i-1}))$;
6: $z_{i,j} \leftarrow Q(s_{i,j}, \gamma)$;
7: **end if**
8: **end for**
9: $Rule_i \leftarrow z_{i,1} \cdots, z_{i,2^d-1}$;
10: **end for**
11: $Rule_v \leftarrow \{Rule_{\pi(\phi(1),\gamma)}, \cdots, Rule_{\pi(\phi(b),\gamma)}\}, \gamma$;
12: // Permute the shuffled blocks with nonce γ;

information leakage to recover the plaintexts. To further reduce above leakage, we propose to conduct secure order comparison via encrypted token matching. Such treatment will bring two benefits: (1) during the comparing processing, the cloud only sees different search tokens instead of order condition. Thus, the attacker cannot learn whether two different requests are conducted in the same order. (2) after the comparison, the cloud only knows whether the requested token matches the encrypted rules. It cannot learn the order association between the tokens and encrypted rules. Furthermore, we modify the adopted ORE scheme that enables hiding the position of the first different block via shuffling block ciphertexts. The comparison can still be performed via looking for the permuted block that matches the encrypted rule condition. The detailed building procedure will be conducted in the next paragraph.

B. Encrypted Header Matching

Header matching enables the middlebox to determine whether an encrypted header lies in a range of rules. The detailed building procedure for a given rule $\mathcal{R}$ is presented in Algorithm 1. This procedure is executed at the client-side gateway, and later the encrypted rule filter will be sent to the outsourced middlebox for header matching inspection. The client first parses two endpoints $\{v^1, v^2\}$ from the range-based rules $\mathcal{R}$. Then for each endpoint, say v^1, the gateway encrypts it via our enhanced ORE scheme as follows. It first splits the endpoint's value v^1 into b blocks with equal length d bits. Then the order result cmp is protected with a random mask. Specifically, the sub block j in block i is encrypted as $Q(F_{k1}(v^*_{|i}||cmp(v^*_{|i}, v^1_{|i}), H(v^1_{|i-1})), \gamma)$, where $v^1_{|i}$ is the current block value, $v^1_{|i-1}$ is its prefix and γ is the unique nonce for this header matching rule $\mathcal{R}$. $v^*_{|i}$ is one of the possible values for this bit block. For instance, if the block size d=2, $v^*_{|i}$ has total $2^d = 4$ possible values $\{00, 01, 10, 11\}$. Note that our design drops the "equal" information as shown in line 4 of Algorithm 1. Thus, each encrypted block actually contains $2^d - 1$ entries. After all the blocks have been encrypted, the algorithm first permutes these entries via the secure PRP function ϕ. Then the permuted blocks will be shuffled again by using the PRP function π with the unique nonce γ. Finally, the building algorithm outputs the permuted ciphertext set $\{Rule_{\pi(\phi(1),\gamma)}, \cdots, Rule_{\pi(\phi(b),\gamma)}\}$ with the corresponding nonce γ as the encrypted header matching rule.

Inspection protocol: Given a key generation function $\mathsf{KGen}(1^\mathsf{k})$, the proposed algorithm $\mathsf{Build}_{\mathsf{rule}}$ and order condition $cmp \in \{>, <\}$, the gateway asks the middlebox within encrypted middlebox rulesets to check whether the inbound header string str matches the ruleset $\mathcal{R}$. The detailed header matching inspection is presented as follows:

• INITIALIZATION:

1) *GW* generates a private key $k_1 \leftarrow \mathsf{KGen}(1^\mathsf{k})$, where k is the security parameter, and builds the encrypted rule $Rule_\mathcal{R} \leftarrow \mathsf{Build}_{\mathsf{rule}}(k_1, \mathcal{R})$. The $Rule_\mathcal{R}$ is sent to *MB* to execute header matching inspection.
2) When a packet arrives, *GW* parses the header string str from the packet and splits it into b blocks. For each block $i \in \{1, b\}$, the client generates secure tokens $t_i \leftarrow F_{k1}(str_{|i}||cmp, H(str_{|i-1}))$, where $str_{|i}$ is the block value, $str_{|i-1}$ is the prefix and $\{F, H\}$ are secure PRFs.
3) *GW* permutes the encrypted token set by secure PRP ϕ, and redirects $t_{str} \leftarrow \{t_{\phi(1)}, \cdots, t_{\phi(b)}\}$ for inspection.

• PRE-PROCESSING:

1) For each header matching rule $Rule_\mathcal{R}$, *MB* parses it as $\{\gamma, Rule_{\pi(\phi(1),\gamma)}, \cdots, Rule_{\pi(\phi(b),\gamma)}\} \leftarrow Rule_\mathcal{R}$.
2) *MB* parses the incoming tokens as $\{t_{\phi(1)}, \cdots, t_{\phi(b)}\} \leftarrow t_{str}$. After that, it computes the permuted token set $\{t_{\pi(\phi(1),\gamma)}, \cdots, t_{\pi(\phi(b),\gamma)}\}$ for each rule, where π is secure PRP and γ is the unique nonce for different rules.
3) *MB* holds up the encrypted traffic and calls ORE comparison to perform header matching inspection.

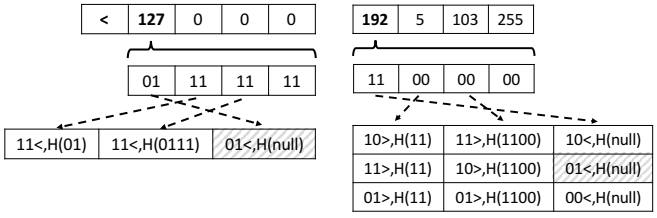
• INSPECTION:

1) *MB* computes $s_i \leftarrow Q(t_{\pi(\phi(i),\gamma)}, \gamma)$ for each permuted token $t_{\pi(\phi(i),\gamma)}$, where $i \in \{1, b\}$. Given $Rule_{\pi(\phi(i),\gamma)}$, *MB* tries to test whether s_i is a member of the set in it.
2) Only if a token s_i related to the header string str is matched, the action is allowed to be triggered.

Illustrative example: The proposed header matching protocol works compatibly with existing firewall filtering as shown in the following range-based rule from an open-source ruleset [22]. This rule will reject all traffic in the blacklist that originates from the source address $192.5.103.0/24$. Our security goal is to protect the rule confidentiality through their life-cycles. To this end, we leverage our proposed ORE scheme to achieve secure header matching over encrypted ruleset.

#Spamhaus: block in log quick from $192.5.103.0/24$ to any

To better understand the inspection protocol of our scheme, Figure 2 illustrates how it works to detect whether the arrived IP address "127.0.0.0" lies in the range of the aforementioned rule $(192.5.103.0, 192.5.103.255)$. Here, we utilize 2-bit ORE



(a) Token generation (b) Header matching ciphertext

Fig. 2: Header matching operation

design to encrypt the right endpoint value and analyze the feasibility of our design in a range-based firewall framework.

As shown in Figure 2(b), the order information $\{>, <\}$ of the first three blocks $\{11, 00, 00\}$ are embedded with their prefix blocks. Then the algorithm 1 outputs these permuted blocks as the encrypted rule of the endpoint "192.5.103.255". Note that all the encrypted entries also need to be masked via PRFs $\{F, Q\}$ with a nonce γ, which is omitted in Figure 2. Regarding the range matching inspection, the *MB* first generates the secure tokens t of "127.0.0.0" based on the order condition " $<$ ", as shown in Figure 2(a). After the same permutation with the unique nonce γ, the middlebox attempts to find the matched block via computing $Q(t, \gamma)$. The result shows that the third token block $F_{k1}(01||<, H(null))$ (after permutation) matches the encrypted endpoint rule. Follow the same treatment, we can find that the left endpoint does not match the request token, and the action will not be executed.

Our header matching design leaks only the equality pattern of the most significant blocks, not the original location of these bit blocks. In addition, it guarantees that the order in each sub block is different, and the order conditions for different values are also different. Besides, our design inherits the merit from [10] that supports prefix-match inspection. Meanwhile, our inspection protocol can also support $\{\geq, \leq\}$ comparison via range extension. Due to the deterministic property of PRF, the range comparison can still correctly be performed via token matching. We note that this is achieved at the cost of leaking the equality of ciphertext after all sub blocks are matched. To minimize this leakage, one straightforward solution is to introduce an extra round of interaction to re-encrypt the matched block, as demonstrated in [23]. In terms of the comparison complexity, our block comparison scheme achieves optimal process time $O(1)$ with low latency, and further support parallel request processing, and good throughput. For each sub block, the worst-case scenario is $2^d - 1$ times operations. Note that there is a tradeoff in efficiency, security and ciphertext space as mentioned in [10]. Specifically, for larger block size d, ciphertext security continues to improve but introducing more matching operation and space cost. Note that the performance can further be improved by conducting token matching of individual blocks in parallel. One of the main limitations of our design is that the length of the ciphertext increases with the block size. To reduce storage overhead, we further customize and integrate a compact version into the encrypted ruleset framework, as illustrated in the next section.

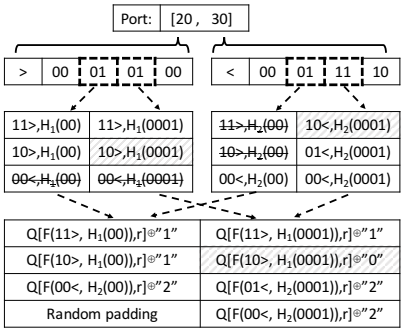


Fig. 3: An example of fast compression for the encrypted filter. Here, the third blocks "01" and "11" have the same prefix "0001", thus they can be compressed into the same encrypted filter. The highlighted entries $F_{k1}(10||>, H_1(0001))$ and $F_{k2}(10||<, H_2(0001))$ will be compressed with the check flag "0", representing that both endpoints match the rule.

C. Compact Header Matching

In this section, we focus on developing a method that specifically addresses the challenges brought up by storage overhead. To address this issue, we propose to fully exploit features of head matching inspecting mechanisms and uniquely integrate it with our secure ORE comparison protocol to reduce space consumption. We refer to such improvement as a rule-aware optimization.

To understand our approach to reducing the space consumption, we first consider what happens if we modify the encryption scheme that only outputs the matched entries as the ciphertexts instead of the all possible values. The header matching inspection can still be conducted via our ORE comparison. Figure 3 shows the building procedure of a port rule, i.e., $[20, 30]$. When we encrypt the second bit block "01" of the left endpoint value 20, the encrypted entry $F_{k1}(00||<, H_1(00))$ will be removed from the result set since it does not match the rule condition " $>$ ". Since this optimized design only stores the rule matched entries, it is desirable to have a space efficient ORE ciphertext. However, designing such an encrypted header matching scheme faces the challenges on achieving both security and efficiency.

First, removing redundant entries has additional leakage that rules with a different number of encrypted entries are distinguishable before inspection. This leakage can be exploited for inference attacks [8], [24] to identify the association between rules and size of ciphertexts. Thus, the encrypted header matching rule should be able to provide constant space consumption. Second, directly applying our basic ORE scheme for the header matching operation does not leverage the full benefits of rule-based inspection. Because our basic design requires to maintain two ORE ciphertexts for individual header matching rule respectively, which incurs considerable burden for storage overhead. Thus, our compact design needs to consider these stringent requirements simultaneously.

To further improve the security and space efficiency, we propose to explore the ORE construction that compresses the same prefix of two endpoints into one ciphertext with constant

Algorithm 2 Build compact header matching rule

Input: Private key k_1; secure PRFs $\{F, Q\}$; secure PRP $\{\pi, \phi\}$; hash function $\{H_1, H_2\}$; endpoints $\{v^1, v^2\}$ with same prefix b blocks; all possible block value v^*.

Output: Compact header matching rule $Rule_{\mathcal{R}}$.

Generate a nonce γ;
for $i \in \{1, b+1\}$ **do** // b prefix and next blocks
  for $j \in \{1, 2^d\}$ **do**
    $s^1_{i,j} \leftarrow F_{k1}(v^*_{|i}|| >, H_1(v^1_{|i-1}))$;
    $s^2_{i,j} \leftarrow F_{k1}(v^*_{|i}|| <, H_2(v^2_{|i-1}))$;
    if $cmp(v^*_{|i}, v^1_{|i})$="$>$"&& $cmp(v^*_{|i}, v^2_{|i})$="$<$" **then**
      $z_{i,j} \leftarrow Q(s^1_{i,j}, \gamma) \oplus$ "0"; // match v^1 and v^2
    else if $cmp(v^*_{|i}, v^1_{|i})$="$>$" **then**
      $z_{i,j} \leftarrow Q(s^1_{i,j}, \gamma) \oplus$ "1"; // match v^1
    else if $cmp(v^*_{|i}, v^2_{|i})$="$<$" **then**
      $z_{i,j} \leftarrow Q(s^2_{i,j}, \gamma) \oplus$ "2"; // match v^2
    else Remove the rest of entries;
    end if
  end for
  Insert random padding;
  // Keep the number of encrypted entries is 2^d
  $Rule_i \leftarrow z_{i,1}, \cdots, z_{i,2^d}$;
end for
$Rule_{\mathcal{R}} \leftarrow \{Rule_{\pi(\phi(1),\gamma)}, \cdots, Rule_{\pi(\phi(b+1),\gamma)}\}, \gamma$;

space complexity to accomplish secure and fast data compression. Such treatment will immediately save the ciphertext space because the duplicates are not required to be separately stored. Meanwhile, the correctness will still be guaranteed due to the one-way property of PRF. Besides, we aware that the encrypted entry is only used for request matching, which is not required to be decrypted. Thus, the space efficiency can further be improved by truncating encrypted entries. The detailed building procedure for the compact header matching rule is presented in Algorithm 2. Given two endpoints' values $\{v^1, v^2\}$ with the same prefix b blocks, building the ith ciphertext block contains the following step:

1) Generate the secure tokens $\{s^1_{i,j}, s^2_{i,j}\}$ for two endpoints, where $s^z_{i,j} = F_{k1}(v^*_{|i}||cmp, H_z(v_{|i-1}))$, $z \in \{1, 2\}$; remove the duplicates including the equality entries.
2) For each block value $v^*_{|i}$, if it matches the order condition of endpoints $\{v^1_{|i}, v^2_{|i}\}$ at the same time, concatenating the token $s^1_{i,j}$ with the nonce γ, and encrypting it by XORing the check flag "0". That is, $z_{i,j} = Q(s^1_{i,j}, \gamma) \oplus$ "0".
3) If the left endpoint's block value $v^1_{|i}$ matches the order condition, the encrypted sub block is $z_{i,j} = Q(s^1_{i,j}, \gamma) \oplus$ "1"; otherwise, encrypts the token $s^2_{i,j}$ with the flag "2".
4) For the empty entries, make them indistinguishable from occupied sub blocks via random padding.

Inspection protocol: Based on the compact header-match framework, the header matching inspection is as follows:

• INITIALIZATION:

1) When a packet arrives, *GW* parses the prefix string str from the packet and splits it into b blocks. For each block i, *GW* generates the prefix token pair $t_i = \{t^1_{i,>}, t^2_{i,<}\}$, where $t^z_{i,cmp} = F_{k1}(str_{|i}||cmp, H_z(str_{|i-1}))$, $z \in \{1, 2\}$.
2) *GW* permutes the encrypted token set by secure PRP ϕ, and sends $t_{str} \leftarrow \{t_{\phi(1)}, \cdots, t_{\phi(b+1)}\}$ to *MB*.

• PRE-PROCESSING:

1) *MB* parses the encrypted header matching rule $Rule_{\mathcal{R}}$ as $\{\gamma, Rule_{\pi(\phi(1),\gamma)}, \cdots, Rule_{\pi(\phi(b+1),\gamma)}\}$.
2) For each incoming token t_{str}, *MB* permutes the token set $\{t_{\pi(\phi(1),\gamma)}, \cdots, t_{\pi(\phi(b+1),\gamma)}\}$ with the nonce γ.
3) *MB* holds up the encrypted traffic and calls ORE comparison to perform header matching inspection.

• INSPECTION:

1) For each block $i \in \{1, b+1\}$, *MB* unmasks the corresponding entry via XORing $Q(t_{\pi(\phi(i),\gamma)}, \gamma)$.
2) Only if the check flags "0" or "1&&2" are revealed (i.e., it means that the token lies in the range of rules), the corresponding action will be triggered.

The resulting rule filter scheme holds the same security guarantee of our basic scheme. Only the encrypted rule size is known to the middlebox. Without inspecting, no other information about the rule content is learned. By encoding the rule-matched entries with check flags to encrypt the same prefix, we do not have to separately store two endpoint ciphertexts. More detailed correctness and security analysis will be shown in the next section.

V. CORRECTNESS AND SECURITY ANALYSIS

In this section, we conduct a rigorous security analysis of the proposed scheme. Then we discuss how our scheme can defend against attacks on encrypted range-based operations.

A. Correctness Analysis

In our design, the core operation of the proposed header matching protocol is to process a given token over the encrypted rule value to obtain the matched entry for a requested range condition. Therefore, to prove our enhanced ORE scheme can return correct header matching results, we need to show that there exists one and only one sub block that matches the order condition during the header matching inspection. Then we give the following theorem:

***Theorem* 1.** *Given a query value m_1 and a targeted value m_2 with b blocks with length of d bits, and the matching condition $\tilde{\times} \leftarrow \{<, >\}$, if $m_1 \ \tilde{\times} \ m_2$ stands, then there exists one and only one matched sub block $s_{i,j}$, where $i \in \{1, b\}$, $j \in \{1, 2^d\}$.*

Proof. Recall that the encrypted entries are generated as $F_{k1}(m_{|i}||cmp, H(m_{|i-1}))$, where $m_{|i}$ is the block value, $m_{|i-1}$ is the prefix and $cmp \in \{>, <\}$. Therefore, if a matched entry is found, m_1 and m_2 must coincide with all the following: 1) the matched blocks should have the same prefix value, i.e., $m_{1_{|i-1}} = m^j_{2_{|i-1}}$; 2) the order condition should be the same; 3) the block value of $m_{1_{|i}}$ should be the same as the sub block value in $m^j_{2_{|i}}$.

Considering $m_{1_{|k}}$ and $m^j_{2_{|k}}$ match the query condition, we assume that there exists another entry $m^P_{2_{|l}}$ in m_2 that matches it as well. Namely, $m_{1_{|k}} = m^j_{2_{|k}} = m^p_{2_{|l}}$, where $k \neq l$ or $j \neq p$. For the case $k \neq l$, it means that the two matched entries are in different blocks. As mentioned, the matched blocks should have the same prefix value. If the two entries are found in two different blocks, their prefix should be different. Thus, this case is untenable. Regarding the case $j \neq p$, it means that the matched entries are in two different sub blocks. Recall that each sub block value in a specific block is unique, therefore $m^j_{2_{|k}} = m^p_{2_{|l}}$ if and only if $j = p$. As a result, there is only one sub block matched the order condition in each inspection. □

B. Security Analysis

Regarding security, our design is built on top of the ORE scheme proposed in [10]. It inherits the advantage of ORE; that is, ORE ciphertext achieves semantic security. No useful information can be learned from the ciphertext only. Besides, our design carefully leverages random permutation and remove equality information from the ciphertexts. The resulting improvement is the first different blocks are protected, as demonstrated in [13]. Following the security framework for search over encrypted data [25], we present rigorous security analysis to demonstrate that our design achieves strong protection on the rule content and client privacy. Meanwhile, we discuss how our design can address existing leakage-abuse attacks.

Based on the security strength, we precisely capture the controlled leakage and formalize the simulation-based security definition. Specifically, we define the views of an adversary who can gain the access to the middlebox. He will observe the inspection requests, the encrypted header matching rules and the corresponding inspection results. The leakage $\mathcal{L}_1$ for a given rule set R is defined as:

$$\mathcal{L}_1(R) = (|H|, |B|, \{b, d, \gamma_i\}_n)$$

The adversary learns the size of encrypted rules $|H|$ and the bit length of each sub block $|B|$. He also knows the number of blocks b containing d sub blocks. The number of ruleset n and the unique nonce γ_i of the rule i are also obtained by the adversary. When a client requests a rule inspection v, the view of an adversary is defined in the leakage $\mathcal{L}_2$ as:

$$\mathcal{L}_2(v, cmp) = (\{B, flg\}_m, t)$$

For a given token t, the adversary learns the number of the matched sub block m, the ciphertext for each matched entries B and corresponding check flags flg. In addition, we define the leakage $\mathcal{L}_3$ to maintain repeated requests as follow:

$$L_3(\mathbf{I}) = (M_{q\times q})$$

where $\mathbf{I}$ is q number of inspection requests. $M_{q\times q}$ is the symmetric bit matrix that maintains the repeated requests. Each element in the $M_{q\times q}$ is initialized as 0. For $i, j \in [1, q]$, the elements of matrix $M_{i,j}$ and $M_{j,i}$ are equal to 1 if two tokens $t_i = t_j$. Given the above definitions of adversary views, the simulation-based security definition of the proposed scheme is given as follows:

***Definition* 1.** *Let* $\Omega = (\mathsf{KGen}, \mathsf{Build}, \mathsf{Process})$ *be our scheme for secure header matching inspection. Given leakage* $\mathcal{L}_1$, $\mathcal{L}_2$ *and* $\mathcal{L}_3$, *we define the following probabilistic experiments* $\mathbf{Real}_{\mathcal{A}}(k)$ *and* $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$ *with a probabilistic polynomial time (PPT) adversary* $\mathcal{A}$ *and a PPT simulator* $\mathcal{S}$*:*

$\mathbf{Real}_{\mathcal{A}}(k)$*: The* GW *executes* $\mathsf{KGen}(1^{\mathsf{k}})$ *to generate the private key* K. $\mathcal{A}$ *selects a set of rules* $\mathbf{R}$ *and asks the* GW *to build ciphertexts via* Build *(Algorithm* 2*). Then* $\mathcal{A}$ *conducts a polynomial number of* q *requests and asks the* MB *for secure tokens and the inspection results via* Process *(Inspection protocol). Finally,* $\mathcal{A}$ *returns a bit as the output.*

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$*:* $\mathcal{A}$ *selects a set of rules* $\mathbf{R}$*. Then* $\mathcal{S}$ *simulates the rule ciphertexts for* $\mathcal{A}$ *based on* $\mathcal{L}_1$. $\mathcal{A}$ *performs a polynomial number of non-adaptive* q *requests. From* $\mathcal{L}_2$ *and* $\mathcal{L}_3$, $\mathcal{S}$ *returns the simulated tokens and ciphertexts. Finally,* $\mathcal{A}$ *returns a bit as the output.*

Ω *is non-adaptively secure with* ($\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$) *if for all PPT adversaries* $\mathcal{A}$*, there exists a simulator* $\mathcal{S}$ *such that:* $Pr[\mathbf{Real}_{\mathcal{A}}(k) = 1] - Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k) = 1] \leq negl(k)$*, where* $negl(k)$ *is a negligible function in* k*.*

***Theorem* 2.** Ω *is non-adaptively secure with* ($\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$) *if* F, Q, H *are secure PRFs.*

Proof. Given $\mathcal{L}_1$, the simulator $\mathcal{S}$ generates the simulated header matching rule H', which is indistinguishable from the real one H. The sizes of H and H' are identical. For each simulated rule H', it contains $b \times d$ sub block entries with equal length $|B|$ of the ciphertext. The bit length of real sub block B and simulated one is the same. But $\mathcal{S}$ generates random strings for each sub block ciphertext.

From $\mathcal{L}_2$, $\mathcal{S}$ can simulate the first token and the result. On the simulated header matching rule H', $\mathcal{S}$ randomly selects m entries, which are the same as the request over the real one, and assigns the resulting check flags flgs to the simulated entries. Then the mask can be simulated as $:B' = Q(t', \gamma) \oplus flg$, where t' is a random string as the simulated token, and flg is identical to the one in the real header matching rule. In particular, we use random oracles $\{\mathcal{H}_F, \mathcal{H}_H\}$ as PRFs $\{F, H\}$. $\mathcal{S}$ can generate a random string $t' = \mathcal{H}_F(v_{|i}||cmp, \mathcal{H}_H(v_{|i-1}))$ as the simulated token, where $i \in \{1, b\}$. From $\mathcal{L}_3$, $\mathcal{S}$ updates $M_{1,1} = 1$ in a matrix $M_{q\times q}$.

In the subsequent jth requests ($j \in \{2, q\}$), if $\mathcal{L}_3$ indicates that the request appears before, $\mathcal{S}$ will select exactly the same entries and use the same tokens and masks generated before. Meanwhile, it will update the corresponding element in $M_{1\times j}$ and $M_{j\times 1}$ to be "1". Otherwise, $\mathcal{S}$ will simulate the token, the entries, and the resulting check flag flg by following the procedure of the first request via $\mathcal{L}_2$.

Due to the pseudo-randomness of PRF and the semantic security of symmetric encryption, $\mathcal{A}$ cannot distinguish the outputs of the real experiment and the simulated one. □

TABLE I: Space and time cost comparison

Encryption Scheme	Block size	Encryption	Comparison	Ciphertext size	Leakage
Boldyreva *et al.* OPE [7]	-	$\sim$7203.64μs	$\sim$0.72μs	16 bytes	(Hard to quantify)
Chenette *et al.* ORE [9]	1 bit	$\sim$4.12μs	$\sim$0.96μs	16 bytes	Order info., First bit that differs
Lewi *et al.* ORE [10]	4 bits	$\sim$108.96μs	$\sim$0.76μs	385 bytes	Order info., First bit block that differs
Yuan *et al.* ORE [14]	2 bits	$\sim$142.70μs	$\sim$2.22μs	528 bytes	First bit block that matches
	4 bits	$\sim$158.43μs	$\sim$1.83μs	1040 bytes	
	8 bits	$\sim$911.02μs	$\sim$2.15μs	2064 bytes	
Our compact ORE	2 bits	$\sim$149.64μs	$\sim$3.60μs	272 bytes	The matched bit block
	4 bits	$\sim$159.72μs	$\sim$1.75μs	528 bytes	
	8 bits	$\sim$716.33μs	$\sim$5.28μs	1040 bytes	

C. More Discussion

In this sub section, we generalize recent attacks on searchable encryption schemes and discuss how our design can defend against these attacks. Our security analysis shows that our design can provide provable security against attackers with snapshot access to the encrypted data.

Sorting attack: In [8], Naveed *et al.* propose a conceptually simple approach to exploit OPE-encrypted ciphertexts. The attacker utilizes the order relations to map each ciphertext to the element of the plaintext space with the same ranks. We note that our design effectively defends against these attacks because the tokenized rule condition is secure against the attacker to learn the order information. In [11], Grubbs *et al.* propose to abstract the sorting attack as a non-crossing bipartite matching problem to improve attack accuracy. This attack requires the information such as ciphertext frequency and order relations, which are protected in our scheme.

Multi-column attack: Durak *et al.* [12] extend the above attack by leveraging queries among different columns. They use inter-column correlations to obtain more information about the underlying values. We note that their work targets the ORE-encrypted columns in an encrypted database table. It is not applicable to our header inspection because the rule contents are encrypted by using independently picked random nonces. By encoding the order into a random mask, it is infeasible to determine the correlations among different ruleset.

Communication volume attack: In [24], Kellaris *et al.* devise a generic reconstruction attack on encrypted range queries. This attack samples enough well-ordered queries and determines the ciphertext order relations by observing the distribution of the result size. We aware that this attack highly relies on prior knowledge on query distribution, which makes it difficult to be launched in our scenarios. Besides, adding dummy rules can further mitigate the attack.

Access pattern attack: In [26], Lacharite and Minaud show that when the dataset is dense, it can recover the ciphertext without prior knowledge of the distribution. We argue that this attack is not directly applicable to the middlebox rule matching because of its strict assumption. They also provide a common attack approach without the assumption of the dense condition. But it still needs the order information to launch the attack. Meanwhile, we emphasize that the re-build solution can effectively defend the attacker to trace the access pattern leakage, which provides the strong security guarantee against these reconstruction attacks [24], [26].

VI. Experimental Evaluation

A. Prototype Implementation

We implement the design prototype[1] and deploy it at Microsoft Azure for the evaluation. It contains two components: a gateway deployed at the trusted client side and a middlebox for header matching inspection deployed in the public cloud.

Gateway: The gateway is deployed at the client side, which processes cryptographic operations, redirects traffic to middlebox, and builds encrypted firewall rules for header matching inspection. We use OpenSSL (v1.0.2g) for the implementation of cryptographic building blocks, including symmetric-key encryption via AES-128 and pseudo-random function via HMAC-256. We leverage Apache Thrift (v0.9.2) to build a binary protocol, which can reach the maximum bandwidth in practical networking. The gateway component consists of more than 6500 lines of C++ code and it works on Ubuntu Server 16.04, which is deployed on Microsoft Azure for simulation.

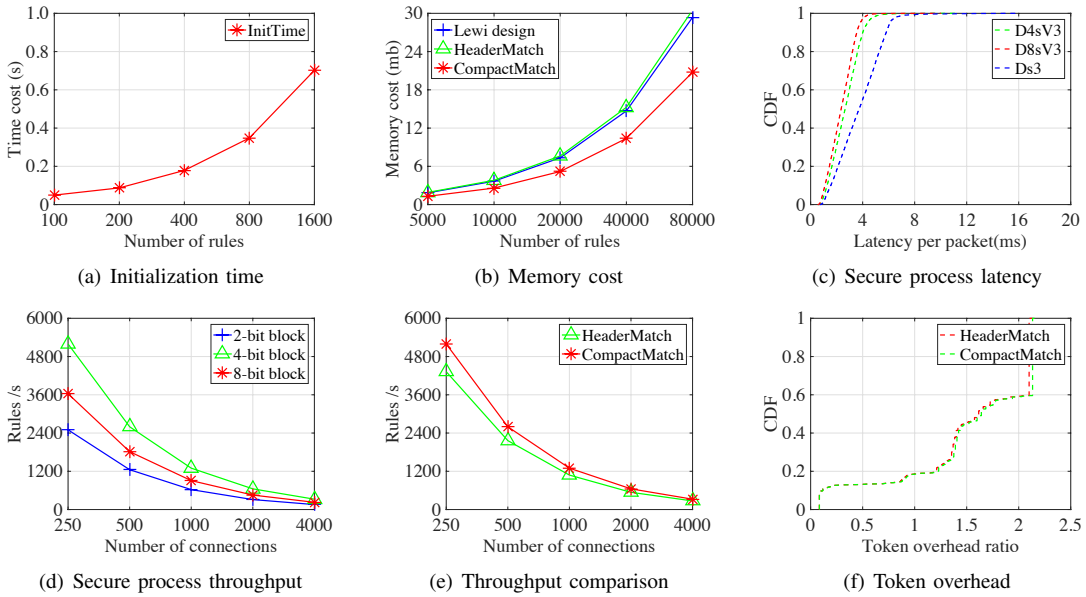
Middlebox: We install the middlebox module to store the encrypted firewall ruleset and perform header matching operation. We develop the rule processing script to enable secure token matching over encrypted ruleset. The implementation consists of more than 6800 lines of C++ code. In this evaluation, we evaluate 2-bit, 4-bit and 8-bit parameter settings for the ORE encryption. To ensure the accuracy of the throughput evaluation, we select a thread-pool server and initialize plenty of threads for processing requests. Thus, the throughput is primarily decided by the computation ability of the middlebox. Meanwhile, the middlebox and the gateway are deployed at the same virtual networking. Therefore, the bandwidth overhead will not be the performance bottleneck in our system.

B. Performance Evaluation

In our experimental evaluation, we target on initialization time, memory cost, processing performance and bandwidth overhead. We also conduct performance comparison between our ORE scheme and existing OPE/ORE schemes.

1) ORE Scheme Evaluation: To use our proposed header matching inspection, the firewall rules need to be encrypted by the encryption algorithm. The detailed performance comparison is shown in Table I. We compare existing OPE/ORE schemes with our compact design (i.e., Section IV-C) by treating the two endpoint values as the firewall rules. As shown in Table I, our compact ORE design is significantly faster than

[1]Prototype online at https://github.com/CongGroup/IWQoS-18.



(a) Initialization time (b) Memory cost (c) Secure process latency

(d) Secure process throughput (e) Throughput comparison (f) Token overhead

Fig. 4: Evaluation for header matching performance

the OPE scheme. For instance, encrypting a single 32 bits firewall rule with our 2-bit ORE design requires just over 149 μs, which is over 48 times faster compared to the Boldyreva *et al.*'s OPE scheme. Of course, when compared with the existing ORE designs without random permutation, our new ORE scheme is much slower. Nonetheless, when using 4-bit ORE design, processing a 32-bit firewall rule requires just 1.75 μs, which is quite modest for practical middlebox inspection. Besides, in exchange, our work leaks strictly less information than the state-of-the-art solutions, as discussed in the security analysis previously. Furthermore, compared with the latest ORE scheme with different parameter settings (i.e., 2, 4 and 8 bits block size), our compact design achieves competitive performance with almost 50% storage savings. According to the observation, our compact ORE design is shown to be capable of providing a flexible balance on data security, space utilization and time efficiency.

2) Header Matching Evaluation: We select real-world firewall rules and traffic dumps to evaluate the performance. The firewall rules come from an open-source ruleset Emerging Threats [22], and the packets are selected from an intrusion detection traffic dump "DARPA"[2], with total over 1.2×10^6 packets. We perform the evaluation on Microsoft Azure. The gateway and middlebox are separately deployed on different models of instances, i.e., "D-series" and "Dv3-series".

Evaluation on encrypted rulesets: We first measure the building performance of the encrypted ruleset with 4-bit ORE construction. Figure 4(a) shows that the time cost grows linearly with the increasing amount of firewall rules. Specifically, the gateway only takes less than 0.75s to finish the build procedure when encrypting 1600 firewall rules. Then we investigate the memory cost of different encryption algorithms in Figure 4(b). Here, we use 2-bit block size for the ORE construction. It shows that the space cost for both our basic design (HeaderMatch) and Lewi *et al.*' ORE scheme follows a similar upward trend as the number of rules increases. As mentioned in [10], there is a tradeoff between space overhead and security. A larger block size has a better security strong while incurring more space cost. Compared with these schemes, our compact header matching design (CompactMatch) only requires one ORE ciphertext for each firewall rule. As shown in Figure 4(b), the size of ciphertext for our compact version is roughly 33% less than these works. When encrypting 5000 firewall rules, the memory cost of our compact design only requires 1.3 MB. This result demonstrates the feasibility of our design on a large scale.

Evaluation on secure request process: The processing latency comes from several aspects including token generation, network transmission, memory access and packets processing. To reduce the influences from gateway and network, we build the gateway using the same instance and conduct the evaluation in the same virtual network. Figure 4(c) depicts the latency CDF of secure header matching process under three different types of instances, i.e., "D8sV3", "D4sV3" and "Ds3". Among them, "D8sV3" and "D4sV3" are the latest instances with a more powerful CPUs (2.4 GHz Intel Xeon® E5-2676 v3 processor), while the "Ds3" belongs to an older previous generation. "D8sV3" has an 8 cores CPU and 32GB RAM comparing to "D4sV3" which is 4 cores with 16GB RAM. As shown in Figure 4(c), the processing latency primarily depends on the CPU frequency rather than the core number or memory size. Specifically, the introduced latency for over 90% of packets is less than 4ms, when using the latest CPU such as "D8sV3" and "D4sV3". Meanwhile, Figure 4(c) also tells that even using the last generation instance, the latency for over 95% of packets is still less than 7ms. The results confirm that our header matching protocol enables fast and effective packet inspection over encrypted firewall rules.

[2]DARPA traffic: online at http://www.ll.mit.edu/ideval/data/

To gain a deeper understanding of our packet processing performance, we further measure the throughput within three different encryption schemes, as shown in Figure 4(d). Recall that our design requires the middlebox server to scan the whole encrypted ruleset. Thus, the time cost increases with the number of added rules. For a "D8sV3" instance with 4-bit encrypted filter design, the throughput for 250 connections can reach up to nearly 5200 rules per second. The overhead comes from the cost of token matching and cryptographic operations during the ORE comparison. Interestingly, for 4-bit block construction, there is an efficiency improvement in header matching performance. For one reason, the comparison complexity for d-bit ORE design is $O(32/d)$. Thus, the throughput of 4-bit scheme is slightly higher than that of 2-bit design. On the other hand, the size of encrypted rule filter increases by almost $2\times$ from 4-bit scheme to 8-bit scheme. Processing a large entry may lead to substantial I/O resources. Overall, there is a tradeoff in data security and inspection performance. The larger block size has stronger security while incurring more performance penalty. Furthermore, Figure 4(e) measures the performance comparison between our basic design and the compact version. Intuitively, our compact design achieves optimal time complexity that locates the matched entries all in a scan. In contrast, the basic design needs to perform ORE comparison on two endpoint values respectively. when holding 500 connections, the inspection throughput of our CompressMatch design can reach up to 2600 rules per second, while that of our basic design is 2100 rules per second. This evaluation result confirms that our design can support secure header matching efficiently.

Evaluation on bandwidth overhead: To enable secure and efficient detection over encrypted traffic, the gateway is required to parse the packet header and generate tokens for inspection. Producing and transmitting those tokens indeed introduce computation and bandwidth overheads. In Figure 4(f), we report the ratio between the header token size and the original packet size. We observe that the introduced bandwidth overhead of both designs for nearly 60% packets is still $2\times$ less than original packet sizes. This result illustrates that our design imposes modest overhead on bandwidth cost.

VII. Conclusion

This paper introduces a middlebox system that can perform encrypted header matching based network functions. We first devise a new ORE scheme that conducts order comparison via token matching without revealing order relations. It also protects partial information of underlying values during comparisons. Then we carefully integrate this cryptographic innovation into a rule-aware size reduction technique to achieve better performance. The prototype is deployed on Azure, and the evaluation results on real-world rulesets confirm the good performance of our design. Our design can be viewed as complementary components to be integrated with systems that support encrypted pattern matching for a more comprehensive and secure outsourced middlebox system.

Acknowledgment

This work was supported in part by the Research Grants Council of Hong Kong under Project CityU 11204215, CityU 11276816, CityU 11212717, and CityU C1008-16G, and the National Natural Science Foundation of China under Project 61572412 and 61672195, and Monash FIT ECR Seed Grant.

References

[1] D. Joseph and I. Stoica, "Modeling middleboxes," *IEEE network*, vol. 22, no. 5, pp. 20–25, 2008.

[2] 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 *Proc. of ACM SIGCOMM*, 2012.

[3] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection for encrypted traffic," in *Proc. of ACM SIGCOMM*, 2015.

[4] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *Proc.of NSDI*, 2016.

[5] X. Yuan, X. Wang, J. Lin, and C. Wang, "Privacy-preserving deep packet inspection in outsourced middleboxes," in *Proc. of INFOCOM*, 2016.

[6] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM SIGCOMM CCR*, vol. 29, no. 4, pp. 147–160, 1999.

[7] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Proc. of CRYPTO*. Springer, 2011.

[8] M. Naveed, S. Kamara, and C. V. Wright, "Inference Attacks on Property-Preserving Encrypted Databases," in *Proc. of ACM CCS*, 2015.

[9] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, "Practical Order-Revealing Encryption with Limited Leakage," in *Proc. of FSE*, 2016.

[10] K. Lewi and D. J. Wu, "Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds," in *Proc. of ACM CCS*, 2016.

[11] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, "Leakage-abuse attacks against order-revealing encryption," in *Proc. of IEEE S&P*, 2017.

[12] F. B. Durak, T. M. DuBuisson, and D. Cash, "What else is revealed by order-revealing encryption?" in *Proc. of ACM CCS*, 2016.

[13] D. Cash, F.-H. Liu, A. O'Neill, and C. Zhang, "Reducing the Leakage in Practical Order-Revealing Encryption," Cryptology ePrint Archive, Report 2016/661, 2016.

[14] X. Yuan, Y. Guo, X. Wang, C. Wang, B. Li, and X. Jia, "Enckv: An encrypted key-value store with rich queries," in *Proc. of ACM AsiaCCS*, 2017.

[15] A. R. Khakpour and A. X. Liu, "First step toward cloud-based firewalling," in *Proc. of IEEE SRDS*, 2012.

[16] J. Shi, Y. Zhang, and S. Zhong, "Privacy-preserving network functionality outsourcing," arXiv preprint arXiv:1502.00389, 2015.

[17] L. Melis, H. Asghar, E. Cristofaro, and M. Kaafar, "Private processing of outsourced network functions: Feasibility and constructions," in *Proc. of ACM Int'l Workshop on Security in SDN & NFV*, 2016.

[18] H. Asghar, L. Melis, C. Soldani, E. Cristofaro, M. Kaafar, and L. Mathy, "Splitbox: Toward efficient private network function virtualization," in *Proc. of ACM HotMiddlebox*, 2016.

[19] C. Wang, X. Yuan, Y. Cui, and K. Ren, "Toward secure outsourced middlebox services: Practices, challenges, and beyond," *IEEE Network*, vol. 32, no. 1, pp. 166–171, 2018.

[20] X. Yuan, H. Duan, and C. Wang, "Bringing execution assurances of pattern matching in outsourced middleboxes," in *Proc. of IEEE ICNP*, 2016.

[21] Y. Guo, C. Wang, and X. Jia, "Enabling secure and dynamic deep packet inspection in outsourced middleboxes," in *Proc. of ACM Int'l workshop on SCC*, 2018.

[22] Emerging Threats, "Emerging threats.net open rulesets." Online at http://rules.emergingthreats.net, 2017.

[23] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. of ACM CCS*, 2017.

[24] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proc. of ACM CCS*, 2016.

[25] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. of ACM CCS*, 2012.

[26] M.-S. Lacharit, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *Proc. of IEEE S&P*, 2018.