Abstract—The explosive growth of network traffic is pushing forward the paradigm of cloud-based middlebox services today. However, due to the increasing attacking surfaces, redirecting enterprises traffic to outsourced middleboxes inevitably raises new privacy concerns about packet content exposure and unauthorized rulesets access. To address these issues, recent efforts have been made towards enabling middlebox services through encrypted traffic and middlebox rules. Following this direction, in this paper, we investigate the issue of privacy-preserving header checking, which is an indispensable service of middlebox applications. Specifically, we propose two new encrypted header-matching schemes that significantly improve security and efficiency. Our main idea is to formulate the problem of encrypted header checking as range-based pattern matching, and carefully craft security designs to enable efficient header inspection in the ciphertext domain. Our first design is carefully tailored to generic range-based functions, while our second design is highly customized for contiguous rule sets to further improve checking efficiency. We formally analyze the security strengths and implement a fully functional system prototype. The extensive experiments over the real-world rule sets demonstrate the practicality of our designs.

Index Terms—Outsourced Middlebox, Order-revealing Encryption, Fuzzy Searchable Encryption, Intrusion Detection

I. INTRODUCTION

Network middlebox has grown to be an essential part of modern networks, which offers a wide variety of network functions, such as firewalls, load balancer, and intrusion detection [1], [2]. Due to the increasing volume of network traffic driven by IoT, managing in-house hardware middleboxes have been regarded as neither economical nor scalable. Therefore, many enterprises have started adopting the public cloud for outsourcing middlebox services [3]–[7]. The related benefits, including reduced capital cost, ease of management, and scalable deployment are well appreciated [8]–[10].

While there are numerous benefits of the cloud-based middlebox service, it also brings new challenging security threats towards the privacy of enterprises. Since the enterprise traffic is redirected to an untrusted environment for packet inspection, the cloud servers could get full access to these sensitive middlebox rule sets and packet headers. Exploiting the rule sets may extract private information of enterprises, e.g., trade secrets and security policy, [11], [12] and analyzing packet headers may derive sensitive information of enterprise infrastructures, e.g., network topology [13]. To improve security, a simple approach is to apply end-to-end encrypted communication protocols like HTTPS, and decrypt the traffic in cloud middleboxes [10], [14] for packet checking. However, this approach would easily constitute man-in-the-middle attacks [15] because the traffic is still processed over middlebox rules in plaintext.

To address the privacy issues, several privacy-preserving middlebox systems [11]–[13], [16] have been proposed that can directly perform packet inspection over encrypted traffic. BlindBox [11] is the first system to enable encrypted packet inspection in cloud-based middleboxes. The follow-up design [12] utilizes encrypted pattern-match schemes to boost efficiency. Unfortunately, neither of them consider the function of range-based firewall filtering, which is one of the fundamental function of modern intrusion detection system. To address this issue, Embark [13] devises a customized prefix-matching scheme to support partial range-based header checking. However, the scheme still has limitations in functionality because firewall filtering can not always be represented by a prefix [1], [17], such as non-contiguous masks 137.98.217.0/8, 22.160.80. Moreover, representing these middlebox rules in prefixes causes a significant increase in the number of ciphertexts [18], which renders inspection performance delayed and storage overhead excess.

Because of the shortcomings of existing solutions, we are motivated to investigate the issue of privacy-preserving yet practical scheme for packet header checking. Our research aim is to enable outsourced middleboxes to detect whether or not a packet header lies in a range of rule values while keeping their information private against the cloud server. To this end, a promising solution to achieve encrypted range queries is allowing the gateway to encrypt its rule sets and packet headers using order-revealing encryption (ORE) [19], and then let the cloud middlebox perform the order comparison over the ORE ciphertexts. ORE [20]–[22] is a recently proposed effective technique for secure range queries, which can transform data values into encrypted bit-blocks and is capable of evaluating order information on ciphertexts. By using ORE, ranges defined in middlebox rules are protected against the attackers even they can access ciphertexts. Recent studies [23], [24] in this field have shown that ORE leaks partial information of plaintext values, i.e., the first differing bit-blocks and order information between two ciphertexts. Thus, directly using the ORE schemes could lead to leakage-abuse attacks [23], [24].
In this paper, we propose two privacy-preserving header checking solutions for outsourced middlebox services. Our design enables the outsourced middleboxes to perform header checking over encrypted rule filters without revealing the sensitive middleboxes rules and the packet headers. We formulate the problem of encrypted header checking as range-based pattern matching and apply our customized cryptographic primitives as the underlying building blocks to enable the checking schemes. Our first solution starts from the practical ORE scheme [21] that enables secure header checking via two rounds of ORE comparison. To protect the order information, we embed orders into random masks for block encryption, and permute these block ciphertexts randomly to protect the original location of the first differing bit-block. Therefore, our enhanced ORE checking scheme overcomes the limitations in the previous solutions.

The above solution can support all types of header checking, but it requires two rounds of comparison for each rule and thus increases the latency. To improve checking efficiency, we also customize a new scheme via wildcard-based encryption for specifically contiguous rulesets. The core idea is to convert the contiguous rule ranges to wildcards and then conduct secure header checking via fuzzy pattern-matching. Thus, our wildcard-based design only needs one-time fuzzy pattern-matching instead of using two rounds of order comparisons. Specifically, we uniquely integrate the recent fuzzy searchable encryption (FSE) scheme [25] with the header checking mechanism to build our secure protocols that achieve optimal efficiency. Our wildcard-based solution encodes packet headers and rulesets into prime-related vectors such that the result of vectors product can be used to determine whether a packet header matches a rule filter or not.

Finally, we analyze our schemes through detailed security analysis and performance evaluation, and the results show that our schemes are secure and feasible. Our proposed schemes provide useful guidelines and new insights on encrypted header checking solutions for middlebox applications. The main contributions of this paper can be summarized as follows:

- We first design a secure header-matching scheme based on our enhanced ORE scheme. It enables the middlebox server to efficiently and accurately match headers and filtering rules while protecting the partial information leaked by prior schemes.
- We further provide an improved wildcard-based construction with optimized checking efficiency. We carefully tailor the designs for broad support of header checking rules with detailed protocols illustration.
- We formulate the formal security definition and prove the correctness and security of our schemes. The comprehensive evaluations on real-world rulesets demonstrate the feasibility and practicability of our designs.

The rest of this paper is organized as follows: Section II presents our system architecture and threat model. Section III describes the details of our header-matching schemes, followed by the security analysis and performance evaluation in Section IV and Section V. Finally, we review the related work in Section VI and conclude the whole paper in Section VII.
**Algorithm 1: Filter\textsubscript{ORE}: Build ORE-based rule filter**

**Input:** primary key $k$; secure PRF $F$; hash functions \{G, H\}; secure PRPs \{\pi, \bar{\pi}\}; bit value $v$; all possible bit values $v'$; orders $cmp \in \{>, <\}$.

**Output:** Encrypted header-match filter $R_0$.

1. Generate a random nonce $\gamma$.
2. Divide bit value $v$ into $b$ blocks with $d$-bits length.
3. For each bit block $\{B_1, ..., B_d\}, i \in \{1, b\}$ do
4.  
5.      For sub-block $\{Z_{i,1}, ..., Z_{i,2^d-1}\}, j \in \{1, 2^d - 1\}$ do
6.         \[ cmp(v'_i, v_i) == \{>, <\} \]
7.         \[ S_{i,j} \leftarrow F_k(v'_i) || cmp(v'_i, v_i) || H(v_{i-1}) \]
8.         \[ Z_{i,j} \leftarrow G(S_{i,j}) \]
9. \[ B_i \leftarrow \{Z_{i,1}, ..., Z_{i,2^d-1}\} \]
10. \[ \{B_{\pi(1)}, ..., B_{\pi(b)}\} \leftarrow \pi[k, \{B_1, ..., B_d\}] \]
11. /* Permute the blocks randomly */
12. \[ R_0 \leftarrow \bar{\pi}[\gamma, \{B_{\pi(1)}, ..., B_{\pi(b)}\}] \]
13. /* Shuffle permuted blocks with $\gamma$ */
14. Output encrypted header-match filter \( R_\gamma \).

**C. Cryptographic Preliminaries**

**Order-revealing encryption:** An order-revealing encryption scheme \{Setup, Enc, Cmp\} is a set of three polynomial time algorithms: The setup algorithm Setup takes a security parameter $\lambda$ as input and outputs a secret key $sk$; the encryption algorithm Enc takes a key $sk$ and a value $v$ as inputs and outputs an ORE ciphertext $ct$; the comparison algorithm Cmp takes two ORE ciphertexts $\{ct_1, ct_2\}$ and outputs a bit $b \in \{0, 1\}$ representing the comparison results. If $b = 1$, it indicates $v_1 < v_2$; if $b = 0$, it indicates $v_1 \geq v_2$.

**Pseudo-random function:** A pseudo-random function $F(\cdot)$ transforms each element $x$ of the set $X$ to an output $y \in Y$ with a secret key $sk \in K$ such that $y$ is computationally indistinguishable from a truly random function. A pseudo-random function $F : X \times K \rightarrow Y$ is $(t, q, \epsilon_F)$ secure if for every oracle algorithm $A$ making at most $q$ oracle queries and with polynomial runtime at most $t$: $|Pr[A_F^{F(k_F)} = 1] - Pr[A^g = 1]| < \epsilon_F$ where $\epsilon_F$ is a negligible function in $F$.

**Pseudo-random permutation:** A pseudo-random permutation $\pi : X \times K \rightarrow X$ is a permutation function on $X$ with all $k \in K$, there is no efficient distinguisher $C$ can distinguish the outputs of $\pi(k, X)$ and the outputs of random permutation function on $X$.

**III. THE PROPOSED SYSTEM**

In this section, we present the design of encrypted header-matching schemes in detail. We first propose a generic design that supports secure header checking based on our enhanced ORE scheme. Our generic design can be seen as a baseline solution that transforms any header rules to range-based pattern matching. We further improve the baseline solution by using the encrypted wildcard-based matching algorithm. The corresponding building procedures and checking protocols are also presented in this section.

**A. Design Rationale**

In practical middlebox applications, the common practice of header checking is detecting whether a packet header matches a range of middlebox rules [26]. To preserve the privacy of headers and rulesets, secure middlebox systems specifically designed for some rule scenarios have already been implemented, such as embark’s PrefixMatch [13] for prefix-matching inspection. However, it still has limitations in functionality [18]. To push forward this area, we exploit recent advancements on ORE schemes [21] to develop secure and generic comparison protocols for range-based firewall filtering. The core idea of ORE scheme is to encode values into bit-blocks so that the first different blocks can be used to reveal the order information. By treating header strings and middlebox rules as data values, it is possible to use ORE schemes to achieve encrypted range-based header checking.

**Enhanced leakage profile:** Although the above-mentioned design has realized all the functional requirements, it does not guarantee strong security protection. The reason is that the existing ORE scheme reveals the location of the first different block as well as order information after comparison. These leakages can be exploited to reveal the exact distances between the plaintext values, which make the scheme suffer from leakage-abuse attacks [22]–[24]. To limit these leakages, our solution is to utilize searchable encryption schemes to encrypt pre-defined comparison results into bit blocks such that the order comparison can be conducted via tokenized block matching. Then, we leverage random permutation over these blocks to hide the original location of the first differing block. With such a privacy-preserving design, the middlebox server can only know whether the encrypted header matches the rule ciphertexts instead of the additional leakages revealed by prior schemes.

**Improved checking efficiency:** The above treatment allows encrypted range-based header checking while simultaneously achieving semantic security. However, it needs to convert rules to independent two endpoint values and conduct two rounds of ORE comparison for range checking, which will lead to possible performance degradation. To improve checking efficiency, we further customize a wildcard-based scheme for the contiguous rulesets. The main idea is to convert rule ranges to wildcards such that only one round of fuzzy search is required to perform range matching. To this end, we resort to a very recent work that achieves accurate and secure fuzzy matching [25], and it is also the most efficient FSE scheme currently. Specifically, we encode the block values of rules and headers as a prime-based vector. Due to the indecomposable property of primes, the inner product of two vectors can be used to evaluate the similarity of headers and rule filters, i.e., the packet headers match the range of rulesets. The detailed building procedures and inspection protocols will be conducted in the next section.

**B. ORE-based Construction**

The detailed building procedure of ORE-based rule filters 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 cloud middlebox for checking headers. Specifically, the gateway first transforms each ruleset to two endpoint values, and encrypts each endpoint value \( v \) via our enhanced ORE scheme. For each bit value \( v \), the gateway divides it into block \( B_i \) with equal length \( d \) bits, and encrypts the comparison results \( cmp \) with a random mask. The sub-block \( Z_{i,j} \) in block \( B_i \) is encrypted as \( G(S_{i,j} || \gamma) \), where \( \gamma \) is the unique nonce and \( S_{i,j} \) is \( F_k(v_i^* || cmp(v_i^*, v_i)) \). \( v_i^* \) is one of the possible values for this block. For instance, if \( d = 2 \), \( v_i^* \) has total \( 2^d \) possible bit values, i.e., \{00, 01, 10, 11\}. Note that our design protects the equality information, as shown in line 4 of Algorithm 1, that the same block actually contains \( 2^d - 1 \) sub-blocks. After all the blocks have been encrypted, the algorithm permutes these blocks via the secure PRP function \( \pi \). Then, the permuted blocks will be shuffled again by using the PRP function \( \tilde{\pi} \) with the unique nonce \( \gamma \). Finally, the algorithm outputs \( \{R_1, \gamma\} \) as the encrypted rule.

**ORE-based checking protocol:** Based on the ORE scheme in Algorithm 1, the gateway asks the cloud middlebox to detect whether the incoming header string \( str \) matches encrypted rulesets. The detailed protocol \( Detect_{ORE} \) of ORE-based inspection is presented as follows:

- **Initialization:**
  1. To outsource the ruleset to the middlebox, the gateway first needs to extract the header strings from the ruleset. Then, each header string is transformed to two endpoint bit values \( \{v_1, v_2\} \).
  2. For each endpoint pair \( \{v_1, v_2\} \), the gateway uses the private key \( k \) to build encrypted filters \( \{R_{v_1}, R_{v_2}, \gamma\} \) by running the algorithm \( Filter_{ORE}(k, v) \). After that, the encrypted filters are sent to the cloud middlebox for executing secure header-matching inspection.

- **Pre-processing:**
  1. When a packet arrives, the gateway needs to generate the header tokens. It extracts the header string \( str \) from the packet and splits it into \( b \) blocks with equal length.
  2. For each block \( i \in \{1, b\} \), the client generates tokens \( t_i \leftarrow F_k(str_i || cmp || H(str_{i-1})) \), where \( str_i \) is the \( i \)-th block value, \( str_{i-1} \) is its prefix value, and \( \gamma \) is the order condition.
  3. The gateway permutes the encrypted tokens \( \{t_1, ..., t_b\} \) by secure PRP \( \pi \), and redirects \( t_{str} \leftarrow \{t_{\pi(1)}, ..., t_{\pi(b)}\} \) to the cloud middlebox for header-matching inspection.

- **Inspection:**
  1. For each encrypted filter value \( R_v \), the middlebox parses it as \( \{B_{\pi(1)}, ..., B_{\pi(b)}\} \).
  2. The middlebox parses tokens as \( \{t_1, ..., t_b\} \sim t_{str} \). After that, it permutes these tokens with the nonce \( \gamma \), i.e., \( t_{\pi(1)}, ..., t_{\pi(b)} \leftarrow \bar{\pi}(\gamma, t_{\pi(1)}, ..., t_{\pi(b)}) \).
  3. For each permuted token \( t_{\pi(i)} \), the middlebox computes \( Z_i \leftarrow G(t_{\pi(i)} || \gamma) \), where \( i \in \{1, b\} \). Then, the middlebox tries to test whether \( Z_i \) is a member of the ruleset \( B_{\pi(i)} \).
  4. Only if both the header tokens \( t_{str} \) match filter values \( \{R_{v_1}, R_{v_2}\} \), the action is allowed to be triggered.

**Protocol instantiation:** Our ORE-based inspection protocol works compatibly with existing firewall filtering as shown in the following rule from an open-source ruleset [27].

This rule will reject all traffic that originates from the non-contiguous source address \([101.192.0.0, 101.195.255.255]\). To better understand the inspection protocol of our scheme, Fig. 2 illustrates how it works to detect whether the arrived packet header “101.193.0.0” lies in the range of the aforementioned rule \([101.192.0.0, 101.195.255.255]\). Here, we take the ORE ciphertext \( T_k(193) \) and rules \( \{R_{t_L(192)}, R_{R_L(195)}\} \) as examples to analyze the feasibility of our ORE-based design. As shown in Fig. 2(b), the pre-defined comparison result for bit blocks \{11, 00, 00, 00\} are embedded with their prefixes. After the random permutation, the algorithm 1 outputs these permuted blocks \( \{B_{L}(00), B_{L}(00), B_{L}(11), B_{L}(00)\} \) as the ORE ciphertext of value “192”. Note that all the encrypted entries also need to be masked via PRFs \( \{F, G\} \) with a nonce \( \gamma \), which is omitted in Fig. 2. During the procedure of header checking, the gateway first generates the header tokens \( \{T_L(01), T_L(00), T_L(11), T_L(01)\} \) based on the block values \{11, 00, 00, 01\} and the order condition “>”, as shown in Fig. 2(a). After the same permutation with the nonce \( \gamma \), the middlebox conducts a secure comparison between the encrypted endpoint and the token. The result shows that the highlighted block of header token \( T_L(01) \sim \gamma \) matches the ORE ciphertext \( B_{L}(00) \), i.e., “01>00”. Following the same treatment (as shown in Fig. 2(c) and (d)), we can find that the packet header “101.193.0.0” lies in the range of the rule \([101.192.0.0, 101.195.255.255]\).

**C. Wildcard-based Construction**

In this subsection, we focus on developing a new construction that improves header checking performance. We note that the ORE-based design requires two rounds of ORE comparison for range checking, which incurs additional computation cost. To improve checking efficiency, we propose to formulate the secure header-matching problem as an encrypted fuzzy search problem. The objective is finding the matched header against a set of predefined fuzzy patterns. Then, we propose an efficient wildcard-based solution by exploiting the indecomposable property of primes and special features of header-matching rule filters. We refer to such a rule-aware optimization as a wildcard-based design.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F, G, H )</td>
<td>pseudo-random functions</td>
</tr>
<tr>
<td>( \pi, \bar{\pi} )</td>
<td>pseudo-random permutations</td>
</tr>
<tr>
<td>( v_i )</td>
<td>bit value of the ( i )-th block</td>
</tr>
<tr>
<td>( v_{i-1} )</td>
<td>prefix value of the ( i )-th block</td>
</tr>
<tr>
<td>( v_i^* )</td>
<td>all possible bit values of the ( i )-th block</td>
</tr>
<tr>
<td>( cmp(v_i^*, v_i) )</td>
<td>comparison result between ( v_i^* ) and ( v_i )</td>
</tr>
<tr>
<td>([,] )</td>
<td>concatenation, wildcard</td>
</tr>
</tbody>
</table>
Algorithm 2: $\text{Filter}_{\text{FSE}}$: Build wildcard-based filter

\begin{algorithmic}
   \State \textbf{Input:} primary key $k$; secure PRF $F$; hash function $H$; secure PRP $\pi$; selected prime $\bar{v}$; header value $v$.
   \State \textbf{Output:} Encrypted header-match filter $R_v$.
   \State 1 Generate a nonce $\gamma$, $t \leftarrow F(k, v_1)$;
   \State 2 Initialize a vector $Z = [Z_1, ..., Z_b]$ with value 1;
   \State 3 for each block value $\{v_1, ..., v_4\}, j \in \{1, 4\}$ do
   \State 4 \hspace{1em} if $v_j \neq \ast$ then
   \State 5 \hspace{2em} $q \leftarrow \pi(v_j || \bar{v}_j)$, $Z_q \leftarrow \bar{v}_j$;
   \State 6 \hspace{1em} else
   \State 7 \hspace{2em} Set $Z = [Z_1 \times \bar{v}_j, ..., Z_b \times \bar{v}_j]$;
   \State 8 \hspace{2em} /* Map each prime to the vector item */
   \State 9 \hspace{2em} for each vector item $\{Z_1, ..., Z_b\}, i \in \{1, b\}$ do
   \State 10 \hspace{3em} $B_i \leftarrow Z_i \oplus H(t||i||\gamma)$;
   \State 11 \hspace{2em} /* Protect vector with random masks */
   \State 12 \hspace{2em} $R_v \leftarrow \{B_1, ..., B_b\}$;
   \State 13 \Output encrypted header-match filter $\{R_v, \gamma\}$;
\end{algorithmic}

Algorithm 2 presents the building procedures of our wildcard-based design. For each contiguous rule (which is in the format as $/8, /16, /24)$, the gateway first splits it into 4 blocks $\{v_1, ..., v_4\}$ and uses the wildcard “$\ast$” to represent the fuzzy part. Then, the gateway selects 4 different primes $\{\bar{v}_1, ..., \bar{v}_4\}$ to delegate each block values. To encrypt the rule filter, the gateway initializes a b-dimensional vector $Z$ and maps each prime into the vector, as shown in Line 3 to 7 in Algorithm 2. Specifically, if the block $v_j$ is not a wildcard, the corresponding prime $\bar{v}_j$ for this block is mapped to the $q$-th vector item, where $q \leftarrow \pi(v_j || \bar{v}_j)$. Otherwise, the gateway computes $\{Z_1 \times \bar{v}_j, ..., Z_b \times \bar{v}_j\}$. After all the primes are mapped to the vector, the gateway embeds each vector item $Z_i$ into a random mask for the vector encryption, i.e., $Z_i \oplus H(t||i||\gamma)$, where $H$ is the hash function, $t$ is the header token that is generated from $v_1$, and $\gamma$ is a unique nonce for this header rule. Finally, the algorithm outputs the encrypted rule filter $\{R_v, \gamma\}$.

Wildcard-based checking protocol: The protocol of header-matching inspection $\text{Detect}_{\text{FSE}}$ based on the scheme in Algorithm 2 is presented as follows:

- **Initialization:**
  1) The gateway extracts the header value $v$ from the ruleset. Each header value is divided into 4 blocks $\{v_1, ..., v_4\}$ and each fuzzy block is represent by the wildcard “$\ast$”.
  2) Given block values $\{v_1, ..., v_4\}$ and the corresponding primes $\{\bar{v}_1, ..., \bar{v}_4\}$, the gateway executes the algorithm $\text{Filter}_{\text{FSE}}(k, v)$ to generate the encrypted rule filter $\{R_v, \gamma\}$. When the encryption operation is completed, the encrypted rule filters will be sent to the cloud middlebox for header-matching detection.

- **Pre-processing:**
  1) The gateway creates a $b$-dimensional vector $\{T_1, ..., T_b\}$ with value “0” to store the header tokens.
  2) After receiving a packet from the external network, it extracts the header string $str$ and divides it into 4 blocks $\{str_1, ..., str_4\}$.
  3) For each block $str_i$, the reciprocal of the $i$-th prime $\bar{v}_i^{-1}$ is mapped to the $q$-th token vector item $T_q$, where $q \leftarrow \pi(str_i || \bar{v}_i)$.
  4) The gateway generates the token $t \leftarrow F(k, v_1)$, and redirects the transformed vector $\{T_1, ..., T_b\}$ together with
t to the cloud middlebox for header-match inspection.

- **Inspection:**
  1. For each encrypted filter value $R_v$, the middlebox parses it as $\{B_1, ..., B_b\}, \gamma$.
  2. For each block cipher $B_i$, the middlebox unmask it via computing $Z_i \leftarrow B_i \oplus H(t|||\gamma)$. Then, the middlebox computes the inner product of two vectors $[Z_1, ..., Z_b]$ and $[T_1, ..., T_b]$.
  3. Only if the results of the inner product is an integer, the corresponding action will be triggered.

**Protocol instantiation:** Based on the building procedures in Algorithm 2, only when the input header is generated from suspicious strings, the middlebox can perform the corresponding action. To understand the protocol of our wildcard-based design, we use a real-world rule item from [27] to illustrate whether the input header “141.101.132.250” lies in the range of the rule filter (141.101.132.0, 141.101.132.255). In this example, we omit the random masks of encrypted filters for clarity of presentation.

**Spanhaus rule:** Block in log quick from 141.101.132.0/24

As shown in Fig. 3, the gateway randomly selects primes $\{7, 11, 13, 17\}$ to represent the rule strings, and maps these primes to the filter vector based on the results of random permutation. In this example, the last block of the rule filter is the wildcard “*”, so that the corresponding prime “17” would be multiplied by each item in this filter vector. After receiving a packet from an external network, the gateway first generates the token vector based on the reciprocal of these primes, and redirects it to the cloud middlebox. Finally, the middlebox conducts the inner product of the token and filter vectors, and the result is an integer (i.e., 52). It means that the input header matches the rule filter and the action, such as dropping the packet, should be executed.

IV. CORRECTNESS AND SECURITY ANALYSIS

In this section, we conduct a formal security analysis for our proposed schemes. We first define the leakage during the inspection protocols, and quantify the security guarantees following the adopted primitives. In addition, we provide a correctness analysis for our enhanced ORE scheme.

A. Correctness Analysis

The correctness of our ORE-based design is guaranteed by the deterministic property of PRF. Explicitly, we prove that there is only one possible sub-block matches the order condition during the ORE comparison. Accordingly, we present the following theorem:

**Theorem 1.** Given two values $v$ and $v^*$ with $b$ blocks, and the condition $\text{cmp} \in \{>, <\}$, if $\text{cmp}(v, v^*)$ stands, there exists one and only one matched sub-block, where $i \in \{1, b\}$.

**Proof.** Recall that the encrypted sub-block is generated as $F_k(v_i^* ||\text{cmp}(v_i^*, v_j)||H(v_{i-1}))$, where $v_i$, $v_i^*$ are block values and $v_{i-1}$ is the prefix. If there is a matched sub-block, $v$ and $v^*$ must coincide with all the following conditions: 1) the same block value $v_i = v_i^*$; 2) the same prefix value $v_{i-1} = v_{i-1}^*$; and 3) the order condition should be the same. Assuming there are two different sub-blocks $\{v_i, v_j\}$ in value $v$ match the order condition of $v_i^*$: Namely, $\{v_i, v_j\}$ are located in two different blocks. As mentioned, the matched sub-block should have the same prefix value. If the two sub-blocks are found in different blocks, the bit length of their prefix values must be different. It means that this assumption is untenable. Therefore, there exists one and only one matched sub-block when $\text{cmp}(v, v^*)$.

B. Security on ORE-based Inspection

Our design inherits the advantage of ORE scheme [21], which achieves semantic security. Besides, we leverage the secure PRF and random permutation to protect the comparison result and the position of the first different bit block, as shown in [22]. Following the security framework of searchable encryption [28], we present rigorous security analysis to demonstrate that our design provides strong protection on the rule filter and client privacy. Specifically, we first define the setup leakage $L_{\text{set}}$ as:

$$L_{\text{set}} = (|Z|, \{b, d, \gamma\})_n$$

where $|R|$ is the size of encrypted rule filter, $b$ and $d$ are the number of blocks and the corresponding bit lengths, $\gamma$ is the unique nonce for the $i$-th rule filter, and $n$ is the number of the rule filter. When the gateway sends an inspection request, the view of an adversary is defined as:

$$L_{\text{arch}} = (t, \{Z_i\})_m$$

where $t$ is the header token, and $\{Z_i\}_m$ is the corresponding $m$ matched sub-blocks. Apart from these leakages, we define the leakage $L_{\text{rpt}}$ to maintain repeat requests:

$$L_{\text{rpt}} = (M_{q \times q}, R_q)$$

where $M_{q \times q}$ is the symmetric bit matrix that tracks repeated $q$ requests and the result set $R_q$. 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 leakages, we provide the simulation-based security definition as follows:
Definition 1. Let $\Omega_{ore} = (KGen, Filter_{ORE}, Detect_{ORE})$ be protocols of ORE-based inspection. Given leakages $\{L^{ORE}_{\text{stp}}, L^{ORE}_{\text{srch}}, L^{ORE}_{\text{rpt}}\}$, we define the following probabilistic experiments $\text{Real}_{A}(k)$ and $\text{Ideal}_{A,S}(k)$ with a probabilistic polynomial time (PPT) adversary $A$ and a PPT simulator $S$:

$\text{Real}_{A}(k)$: The gateway executes $KGen$ to generate the private key $K$. $A$ selects a set of rules $R$ and asks the gateway to build the encrypted ruleset via the $Filter_{ORE}$ algorithm. Then $A$ conducts a polynomial number of $q$ requests and executes the $Detect_{ORE}$ algorithm. Finally, $A$ returns a bit as the output.

$\text{Ideal}_{A,S}(k)$: $A$ selects a set of rules $R$, and $S$ simulates the encrypted ruleset based on $L^{ORE}_{\text{stp}}$. Then $A$ performs a polynomial number of $q$ requests. From leakages $L^{ORE}_{\text{srch}}$ and $L^{ORE}_{\text{rpt}}$, $S$ returns the simulated tokens. Finally, $A$ returns a bit as the output.

$\Omega_{ore}$ is non-adaptively secure with $\{L^{ORE}_{\text{stp}}, L^{ORE}_{\text{srch}}, L^{ORE}_{\text{rpt}}\}$ if for all PPT adversaries $A$, there exists a simulator $S$ such that: $Pr[\text{Real}_{A}(k) = 1] - Pr[\text{Ideal}_{A,S}(k) = 1] \leq \text{negl}(k)$, where $\text{negl}(k)$ is a negligible function in $k$.

Theorem 2. $\Omega_{ore}$ is non-adaptively secure with the leakages $\{L^{ORE}_{\text{stp}}, L^{ORE}_{\text{srch}}, L^{ORE}_{\text{rpt}}\}$ under the random-oracle model if $\{G, H, F\}$ are secure PRFs.

Proof. Given $L^{ORE}_{\text{srch}}$, $S$ can simulate an indistinguishable ruleset $R$. It contains $n$ random items, where each item contains $b \times (2^{v} - 1)$ random strings with equal length to the real one. From $L^{ORE}_{\text{srch}}$, $S$ can simulate the header token and the result set. It randomly selects $m$ entries, which are the same as the results over the real one. Each sub-block can be simulated as $G(T, \gamma)$, where $T$ is the simulated token and $\gamma$ is a random string. In particular, $S$ splits each header $str$ into $b$ blocks and generates $T = F^b(str_i||cmp)||H^b(str_{i-1})$ as the simulated token, where $i \in \{1, b\}$ and $\{F^b, H^b\}$ are random oracles. From $L^{ORE}_{\text{rpt}}$, $S$ updates $M_{1,1} = 1$ in a matrix $M_{q \times q}$. For the subsequent queries, if $L^{ORE}_{\text{rpt}}$ indicates the request appearing before, $S$ will select exactly the same items and use the same tokens generated before. Otherwise, $S$ will simulate the token and result set by following the procedure via $L^{ORE}_{\text{srch}}$. Due to the pseudo-randomness of PRF and the semantic security of symmetric encryption, the adversary $A$ cannot distinguish the outputs of the real experiment $\text{Real}_{A}(k)$ and the simulated experiment $\text{Ideal}_{A,S}(k)$.

C. Security on Wildcard-based Inspection

Our wildcard-based design is built on the scheme of encrypted fuzzy search [25], which provisions the cloud middlebox a controlled capability to search over encrypted data. Once the gateway uploads the encrypted rule filter to the cloud, the size of it will be learned. During the header inspection, the search pattern and the access pattern will be revealed, where the search pattern is the repeated header token and the access pattern indicates the matched rule filters. In this subsection, we follow the security notion of searchable encryption to provide a formal security analysis for our design. We first define the setup leakage $L_{\text{stp}}^{fse}$ as:

$L_{\text{stp}}^{fse} = (|Z|, b, \{\gamma_i\}_n)$

where $|Z|$ is the bit length of each vector item, $b$ is the dimensional of the filter vector, $\gamma_i$ is the unique nonce for the $i$-th rule filter, and $n$ is the total number of the ruleset.

$L_{\text{srch}}^{fse} = (t, \{R\}_m)$

where $t$ is the encrypted header token, and $\{R\}_m$ is the $m$ matched rule filter.

$L_{\text{rpt}}^{fse} = (M_{q \times q}, R_q)$

where $M_{q \times q}$ is the symmetric bit matrix that maintains the repeated $q$ requests and $R_q$ is the result set. Given the above definitions of adversary views, the security definition of our wildcard-based solution is given as follows:

Definition 2. Let $\Omega_{fse} = (KGen, Filter_{FSE}, Detect_{FSE})$ be the protocols of wildcard-based inspection. Given leakages $\{L^{fse}_{\text{stp}}, L^{fse}_{\text{srch}}, L^{fse}_{\text{rpt}}\}$, we define the following probabilistic experiments $\text{Real}_{A}(k)$ and $\text{Ideal}_{A,S}(k)$ with a probabilistic polynomial time (PPT) adversary $A$ and a PPT simulator $S$:

$\text{Real}_{A}(k)$: The gateway calls $KGen$ to get a private key $K$. $A$ selects a set of rules $R$ and asks the gateway to generate the encrypted ruleset via the $Filter_{FSE}$ algorithm. Then $A$ conducts a polynomial number of $q$ requests with tokens and performs the $Detect_{FSE}$ algorithm. Finally, $A$ returns a bit as the output.

$\text{Ideal}_{A,S}(k)$: Given the ruleset $R$ from $A$, $S$ simulates the encrypted ruleset based on $L^{fse}_{\text{stp}}$. Then $A$ performs a polynomial number of $q$ requests. From leakages $L^{fse}_{\text{srch}}$ and $L^{fse}_{\text{rpt}}$, $S$ returns the simulated tokens. Finally, $A$ returns a bit as the output.

$\Omega_{fse}$ is non-adaptively secure with $\{L^{fse}_{\text{stp}}, L^{fse}_{\text{srch}}, L^{fse}_{\text{rpt}}\}$ if for all PPT adversaries $A$, there exists a simulator $S$ such that: $Pr[\text{Real}_{A}(k) = 1] - Pr[\text{Ideal}_{A,S}(k) = 1] \leq \text{negl}(k)$, where $\text{negl}(k)$ is a negligible function in $k$.

Theorem 3. $\Omega_{fse}$ is non-adaptively secure with the leakages $\{L^{fse}_{\text{stp}}, L^{fse}_{\text{srch}}, L^{fse}_{\text{rpt}}\}$ under the random-oracle model if $\{G, H, F\}$ are secure PRFs.

Proof. From $L^{fse}_{\text{srch}}$, the simulator $S$ simulates the encrypted rule filters, which have the same size as the real one. Each filter contains $b$ items, where the content is a $|Z|$-bit random string. From $L^{fse}_{\text{srch}}$, $S$ can simulate the header token and the result. It randomly selects $m$ filters, which are the same as the results over the real one. Each entry in the filter can be simulated as $\bar{v} \oplus H^b(t||\gamma)$, where $H^b$ is an oracle, $\bar{v}$ is a random prime, and $\{t, \gamma\}$ are two random strings as the simulated token and the nonce. And from $L^{fse}_{\text{rpt}}$, $S$ updates $M_{1,1} = 1$ in a matrix $M_{q \times q}$. In the subsequent $i$-th request, if the request $q$ appears repeatedly, $S$ will choose the same result $R_q$ and use the same tokens generated before. Meanwhile, it will update the corresponding elements in $M_{1,i}$ and $M_{i,1}$ to be 1. Otherwise, $S$ will simulate tokens and operate random oracle to get the results as shown in the first query procedure. Due to the semantic security of secure PRF, $A$ cannot differentiate the simulated scheme $\text{Ideal}_{A,S}(k)$ and the real scheme $\text{Real}_{A}(k)$.
D. More Discussion

In this section, we generalize recent attacks on searchable encryption and discuss how our design can defend against these attacks. The security analysis confirms that our enhanced ORE scheme can provide provable security against attackers with snapshot access to the encrypted data.

**Against sorting attacks:** The sorting attacks [23], [24], [29] mainly depend on the order information and the statistical information of the plaintexts to carry out the attacks. It exploits the order relations to map each ciphertext to the element of the plaintext space with the same ranks. The first sorting attack was proposed by Naveed et al. [29]. Later in [24], Grubbs et al. proposed to abstract the sorting attack as the problem of non-crossing bipartite matching to improve attack accuracy.

It is worth noting that the sorting attacks relying on the leakage of orders against the deterministic OPE schemes. To resist these attacks, our proposed encryption scheme embeds the tokenized order information into random masks and conducts order comparison via token matching. By encoding the orders into random masks, it is infeasible to determine the correlations among different rule ciphertexts.

**Against MSDB leakage attacks:** The MSDB leakage based attacks target on specific ORE schemes [20], [21] with specific leaky information, i.e., orders of underlying ciphertexts and the first different bit of ciphertexts. Since the leakage profiles contain the position of the first different bit of any two plaintexts and the order of any two plaintexts, the attacker can directly infer the values of each different bit after multiple comparisons. In [23], Durak et al. proposed to infer the plaintext by computing the distance minimization. Grubbs et al. [24] exploited the leakage profile to narrow down the mapping set and applied the non-crossing attack to match the ciphertexts with auxiliary plaintexts.

We note that this attack can hardly determine the plaintext in our ORE construction because bit values are encrypted into ciphertext blocks. Besides, both the token blocks and ORE ciphertexts are randomly shuffled before sending to the cloud middlebox, the attacker cannot learn the original location of these permuted blocks even after comparison. As a result, our enhanced ORE scheme provides the strong security guarantee against these attacks.

V. Experimental Evaluation

A. Prototype implementation

To evaluate the performance of our designs, we implement the system prototype\(^1\) of the gateway and middlebox modules in C++, and a rule parser in C#. Then we conduct the evaluation on a computing machine with an Intel(R) Core(TM) i7-8700 processor (3.2 GHz) and 8GB RAM. In this experiment, we select real-world traffic dumps and firewall rulesets to evaluate the checking performance. The packets are selected from an intrusion detection traffic dumps “DARPA”\(^2\) with total over 1.2 × 10^6 packets and the firewall rulesets come from an open-source ruleset Emerging Threats [27].

**Gateway:** The gateway module is deployed at the client side, which builds encrypted rulesets, generates header tokens, and redirects tokens to middleboxes for encrypted header checking. We use OpenSSL (v1.0.2g) to implement the cryptographic building blocks, including symmetric-key encryption via AES-128 and pseudo-random function via HMAC-256. The gateway module works on Ubuntu Server 16.04 which consists of around 6500 lines of C++ code.

**Middlebox:** The middlebox module stores encrypted firewall rulesets and executes cryptographic operations of header-matching. We develop the C++ scripts to enable token matching over encrypted rulesets. The implementation of middlebox module consists of more than 6800 lines of C++ code, including around 2400 lines for the proposed encryption schemes. For ORE-based checking evaluation, we use 2-bit, 4-bit and 8-bit parameter settings as the block size for the ORE encryption. To ensure the accuracy of 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.

B. Performance Evaluation

Our experimental evaluation targets on testing the practicality of the proposed header-matching schemes, including initialization time, memory cost, checking efficiency, and bandwidth overhead. We also compare our enhanced encryption scheme with existing OPE/ORE schemes in this section.

**Evaluation on ORE scheme:** To enable generic header checking for all types of rulesets, the firewall rule values need to be encrypted by the ORE scheme in algorithm 1. In Table II, we conduct a comprehensive comparison between our enhanced design and the representative OPE/ORE schemes [20], [21], [30] when using these schemes to generate encrypted rulesets. As shown in Table II, our design is more secure than the state-of-the-art solutions, because it leaks no more information than the access pattern (i.e., the matched cipher block). In contrast, the other schemes are subject to the order leakages in which an attack can infer the plaintexts by monitoring the result distribution. In terms of encryption time, we observe that our design is significantly faster than the OPE scheme. For instance, encrypting a 32 bits rule value with our 4-bit ORE design requires just over 162μs, which is over 44 times faster compared to the Boldyreva et al.’s OPE scheme [30]. For order comparison, our scheme introduces additional computation cost due to the random permutation. Nonetheless, the average processing time is still within an acceptable level. Specifically, when using a 4-bit ORE design for header checking, it just requires about 1.94μs per rule, which is quite modest for practical middlebox applications. According to the observation, our new ORE design is shown to be capable of providing a flexible balance on data security, space utilization, and time efficiency.

**Evaluation on system initialization:** We first test the total time cost of building the encrypted rulesets using algorithm 1 and algorithm 2. And for the better comparison, we implement the encrypted rules with 2-bit, 4-bit, 8-bit setting and measure the performance with the same ruleset. As shown in

\(^1\)Prototype: online at https://github.com/JiaGroup2019/IoT2020

\(^2\)DARPA traffic: online at https://www.ll.mit.edu/r-d/datasets
Figure 4(a) and Figure 4(b), we observe that the time cost of system initialization is in linear to the number of rule items. Specifically, the gateway only takes less than 0.75s to finish the build procedure when encrypting 1600 firewall rules. Specifically, given 1600 rules, our design with 2-bit construction takes about 0.34s in the rule preparation, while the 8-bit design takes about 0.51s. For our optimized wildcard-based solution, the gateway only takes less 0.035s to finish the build procedure when encrypting 1600 firewall rules. We note that this building procedure is a one-time cost.

We then investigate the memory cost of different encryption algorithms in Figure 4(c). Here, we test the memory cost by using 2-bit ORE construction and 20-length FSE vector to generate middlebox rules, respectively. Figure 4(c) shows that the space cost for both our ORE-based design and wildcard-based design grow linearly with the increasing amount of firewall rules. For instance, encrypting 20000 rules with our wildcard-based scheme only requires about 6.1MB. Regarding the space consumption of the ORE-based design, it depends on the bit length of each block. Specifically, our 2-bit design contains $2^{21}$ sub-blocks, where each sub-block is truncated to 64 bits. With a 128-bit random nonce, encrypting two endpoint values for each rule require $(64 \times 3 \times 32/2) \times 2 + 128$ bits. As shown in Figure 4(c), when encrypting 80000 rules, the memory cost of our ORE-based design requires 30MB. The results demonstrates the feasibility of our design for large-scale deployments.

**Evaluation on checking efficiency:** To assess the practicality of our header-matching scheme, we further evaluate the checking latency of encrypted rulesets. The total time consists of the time cost for cryptographic operations at the middlebox server, and the time cost for token generation at the gateway. In particular, Figure 4(d) and Figure 4(e) present the latency CDFs for secure header-matching process under different types of encryption schemes. As shown in Figure 4(d), the average latency for 2-bit and 4-bit ORE constructions range from 5ms to 7ms, which are extremely fast. Meanwhile, Figure 4(d) also tells the introduced latency for over 95% packets is less than 12ms when using the 8-bit ORE scheme. This is because the header checking in the ORE scheme asks the server to perform token matching over encrypted sub-blocks, the 8-bit scheme contains more sub-blocks than the other schemes. Therefore, it is slower, but it still can achieve processing within millisecond latency. In Figure 4(e), we also measure the latency CDF for our optimal wildcard-based solution. It shows that only 2ms latency overhead is introduced for processing over 95% of packet headers, because it only requires one-time fuzzy search operation. As a result, the overhead introduced by our wildcard-based design is much smaller than ORE designs. The results confirm that our header-matching protocols enable fast and effective packet checking over encrypted firewall rules.

To gain a deeper understanding of checking efficiency, we further measure the throughput within three different types of ORE schemes, as shown in Figure 4(f). Recall that our design requires the server to scan the whole encrypted rule sets for header checking. Therefore, the time cost grows linearly with the increasing amount of request connects. Specifically, the throughput of 2-bit ORE design decreases from about 1995 rules/s to 997 rules/s as the number of connects grows from 500 to 1000. And the throughput for 250 connections with 4-bit ORE scheme can reach up to nearly 7653 rules per second, which is approximately 6.3× higher than the throughput of 8-bit scheme. An interesting result shows that 4-bit ORE scheme achieves a better efficiency than other ORE schemes. This is because the comparison complexity for ORE-based header-checking is $O(32/d)$, where $d$ is the block size. Therefore, the throughput of 4-bit scheme is slightly higher than that of 2-bit design. On the other hand, the size of encrypted rule filters increase by almost 2× from 4-bit scheme to 8-bit scheme. Reading a large size ciphertext introduces a significant I/O overhead. Overall, there is a tradeoff in data security and checking performance. A larger block size has stronger security while incurring more performance penalty.

In Figure 4(g), we measure the performance comparison between our ORE-based design and the wildcard-based scheme. Intuitively, our wildcard-based design achieves optimal time complexity by matching rule patterns all in a scan. In contrast, the ORE-based design needs to support range-based header checking by performing two rounds of order comparisons. In particular, when holding 500 connections, the checking throughput of our optimal design can reach up to 3903 rules per second, which is about 2× higher than the throughput of ORE-based scheme. In exchange, our ORE-based scheme can support any range-based firewall rulesets while the wildcard-based scheme can only handle contiguous rulesets. According to the results, our new schemes achieve acceptable checking efficiency and strong security guarantees.

**Evaluation on bandwidth overhead:** In this experiment, we also evaluate the bandwidth overhead by measuring the ratio between the header token and the packet size. As shown in

---

**TABLE II: Comparison of query efficiency and security properties**

<table>
<thead>
<tr>
<th>Encryption Scheme</th>
<th>Block setting</th>
<th>Encryption</th>
<th>Comparison</th>
<th>Ciphertext size</th>
<th>Leakages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boldyreva et al. OPE [30]</td>
<td>-</td>
<td>~7203.64µs</td>
<td>~0.72µs</td>
<td>16 bytes</td>
<td>(Hard to quantify)</td>
</tr>
<tr>
<td>Chenette et al. ORE [20]</td>
<td>1 bit</td>
<td>~4.12µs</td>
<td>~0.96µs</td>
<td>16 bytes</td>
<td>Order leakage, First different bit value</td>
</tr>
<tr>
<td>Lewi et al. ORE [21]</td>
<td>4 bits</td>
<td>~108.96µs</td>
<td>~0.76µs</td>
<td>385 bytes</td>
<td>Order leakage, First different bit-block</td>
</tr>
<tr>
<td>Our enhanced ORE</td>
<td>2 bits</td>
<td>~133.60µs</td>
<td>~2.30µs</td>
<td>400 bytes</td>
<td>The matched bit-block</td>
</tr>
<tr>
<td></td>
<td>4 bits</td>
<td>~162.36µs</td>
<td>~1.94µs</td>
<td>976 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 bits</td>
<td>~921.02µs</td>
<td>~2.20µs</td>
<td>2032 bytes</td>
<td></td>
</tr>
</tbody>
</table>
their security assumption relies on non-colluding servers or trusted hardware on the cloud side.

**Searchable Encryption:** Another line of related works targeted on cryptographic primitives [19]–[21], [25], [43]–[46] for encrypted rich queries, i.e., Order-revealing encryption (ORE) and fuzzy searchable encryption (FSE). ORE is a very useful primitive for cloud servers to make comparisons between ciphertexts as if it had operated on plaintexts [47]–[50]. The initial results, also known as order-preserving encryption (OPE) [30], [51]–[53], only support numerical comparison, while leaking the data distribution [29], [54]. To improve the security of OPE, Chenette et al. in [20] proposed the first practical ORE scheme. The ORE ciphertexts preserve semantic security, while the order relations and the first differing bit are revealed during order comparison. To address this issue, Lewi et al. [21] considered using block-based encryption for hiding the bit value leakages. In a concurrent and independent work, Cash et al. [22] provided another ORE construction based on bilinear pairings to further reduce the leakage above.

The first FSE scheme was proposed by Li et al. [55], which tolerated keyword misspellings in the query. Since then, many follow-up works have been conducted in this research field. The work [56] proposed a symbol-based mechanism to support fuzzy search with edit distance as the similarity metric. In [57], [58], locality-sensitive hashing (LSH) [59] and Bloom filters [60] were used to generated encrypted indexes, but incurring false positives. To improve search accuracy, Fu et al. [61] developed a stemming algorithm to search data with the same prefix. To improve the security, Yuan et al. [62] improved the security of collision counting LSH function by hiding frequency of queries. Ding et al. [63] proposed a random traversal algorithm, which produced different visiting paths on the index for the identical queries with different keys. Very recently, Liu et al. [25] proposed a practical FSE scheme by utilizing the indecomposable property of primes, which is adopted in our design.

**Difference from conference version:** Portions of the work...
presented in this paper have previously appeared in [18]. We have revised the paper a lot and improved many technical details. The primary improvements are summarized as follows: First, we introduce a new wildcard-based scheme that is specifically tailored for contiguous firewall filtering rule sets. In particular, we formulate the problem of encrypted header-matching as secure fuzzy pattern-matching, and devise an inspection design by leveraging the practical FSE scheme [25], as shown in new Section 3.3. The proposed new design can provide better processing performance than our previous schemes. Second, we provide a detailed checking protocol for packet header inspection, and use specific examples to illustrate the feasibility of our solution. Third, we enhance the Section 4 with more comprehensive security analysis of our new scheme. Finally, we redo all the experiments and extend the performance evaluation by using real-world rule sets in Section 5. We also substantially improve the related work in Section 6, which now faithfully reflects recent advancements on a wide range of related topics with clear classification.

VII. CONCLUSION

In this paper, we design and implement secure header checking schemes for outsourced middlebox services. We first formulate the problem as range-based pattern matching, and then propose an ORE-based protocol that can support all generic header checking. Compared with previous designs, our proposed scheme can reduce accessible information and be resistant to the adversaries in the bounded leakage setting. To improve efficiency, we further customize a wildcard-based protocol that specifically serves the contiguous firewall filtering. We provide thorough security analysis to show that our designs can protect the confidentiality of packet headers and inspection rules. The experimental results over real rule sets have shown that our designs achieve practical header checking efficiency. The proposed schemes can be viewed as security components, which can be applied in any range-based network functions for more comprehensive and secure middlebox services.

One interesting direction of future work is to extend our design to other middlebox services such as the secure range-based pattern matching for secure middleboxes, as shown in new Section 3.3. The proposed new design can provide better processing performance than our previous schemes. Second, we provide a detailed checking protocol for packet header inspection, and use specific examples to illustrate the feasibility of our solution. Third, we enhance the Section 4 with more comprehensive security analysis of our new scheme. Finally, we redo all the experiments and extend the performance evaluation by using real-world rule sets in Section 5. We also substantially improve the related work in Section 6, which now faithfully reflects recent advancements on a wide range of related topics with clear classification.

REFERENCES
