Privacy-Preserving Deep Packet Filtering over Encrypted Traffic in Software-Defined Networks

Yi-Hui Lin, Shan-Hsiang Shen, Ming-Hong Yang, De-Nian Yang, and Wen-Tsuen Chen

Institute of Information Science, Academia Sinica, Taipei, Taiwan 115
Email:{yihui1223,sshen3,curtisyang,dnyang,chenwt}@iis.sinica.edu.tw

***Abstract*—Deep packet filtering (DPF) has been demonstrated as an essential technique for effective fine-grained access controls, but it is commonly recognized that the technique may invade the individual privacy of the users. Secure computation can address the tradeoff between privacy and DPF functionality, but the current solutions limit the scalability of the network due to the intensive computation overheads and large connection setup delay, especially for the latest network paradigm, network function virtualisation (NFV) and software-defined network (SDN). In this paper, therefore, we propose a privacy-preserving deep packet filtering protocol, named DPF-ET, that can efficiently perform filtering function over encrypted traffic while diminishing the communication overhead and setup delay for the controller in SDN. DPF-ET guarantees the data privacy for users and remains rule privacy for the network owner. The implementation results on an experimental HP SDN/NFV platform demonstrate that the proposed DPF-ET outperforms the current approaches by reducing 250 times in the communications overhead and 32 times in the setup delay.**

I. Introduction

Deep packet filtering (DPF) plays a crucial role in sophisticated access control for large networks. Through investigating the headers and the payload of the packets in Layer 4-7, network administrators are able to observe and filter traffic flows based on the fine-grained policies. For example, an enterprise network operator can block a file transfer to prevent internal data leaking but allow voice calls between the same pairs of clients [17]. Moreover, an ISP can provide URL filtering over HTTP traffic to block malicious web pages for their customers [18].

However, since DPF is inclined to partially jeopardize the individual privacy [5], end-to-end encryption protocols, such as IPSec and TLS/SSL, are widely adopted by clients to ensure the privacy of data streams. For the encrypted data, prediction and classification of the traffic flows based on machine learning [1][11] are proposed for DPF to maintain the user privacy, but the accuracy is lower than the traditional DPF schemes. Moreover, users still need to trust an authorized proxy to process and analyze their traffic, and it still has a chance to invade user privacy [19].

To address the above issue, secure computation is proposed to balance the trade-off between the user privacy and DPF effectiveness, by allowing the clients to jointly compute a function with their private input data [4]. Private set inclusion, one of the primary functions in secure computation, is able to examine encrypted packets without violating user privacy [13]. However, it usually requires longer computation time, i.e., subseconds for a 1K rule set with 32-bit rule length, and thereby is difficult to scale for large networks. BlindBox [15] is proposed to conquer the above challenge with a tailored protocol for the user and the middlebox with the DPF network function. The packet process time can be improved since it only takes twice 128-bit AES operations.

Recent development on Network Function Virtualisation (NFV) [20] and Software-Defined Network (SDN) [8] is a promising way to support the large-scale deployment of security services (e.g., DDOS defense [6] and Anomaly Detection [9]). As shown in Fig. 1, since the controller has full knowledge of the whole SDN/NFV system, the network operator can first place different numbers of VNFs through the VNF control channels according to the corresponding security policies, and the traffic are then forwarded to the designated VNFs installed with fined-grained rules through OpenFlow [8]. To support the on-line matching of the encrypted data streams and the filtering rules in DPF, it is necessary to first generate the encrypted rules with each user and then install them in VNFs. The SDN controller plays a crucial role during the above procedure to effectively optimize the computation resources of the whole SDN. However, for DPF with BlindBox [15], large communications overhead and connection setup time between the controller and each user tend to be incurred (e.g., 9.5s process time and 500MB transmission size for encrypting a 1K rule set with 32-bit rule length) [15], because BlindBox exploits Yao's garble circuit [16] to securely compute AES encryption for each rule. Therefore, the above properties in traditional DPF are inclined to limit the scalability of DPF in SDNs.

To address the above issues, we design a low-overhead privacy-preserving DPF protocol for SDNs in this paper. The protocol is composed of a setup phase and a process phase. The setup phase generates encrypted filtering rules, while the process phase matches the encrypted rules with encrypted packets from users. To ensure the privacy of users, the network will not learn the data stream if it does not match any filtering rules. On the other hand, the rule privacy (i.e., the rules will not be acquired by any users) is also sustained. More specifically, we prove that the user and rule privacy are both preserved in our DPF protocol according to Canetti's model [4] under random oracle assumption [3]. Moreover, we encrypt the filtering rules and the data streams based on a fast OT extension

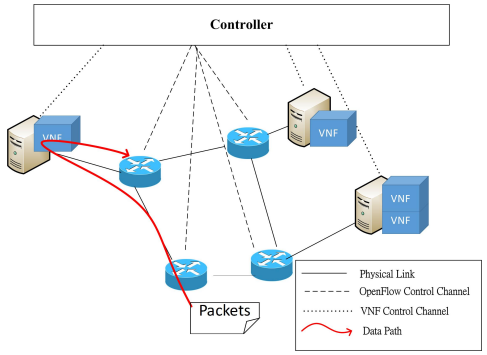

Fig. 1. SDN/NFV Platform: Network traffic is forwarded to VNFs.

[2] to effectively reduce the communications overhead. We implement the proposed protocol on HP SDN/NFV platform, and the experimental results manifest that the protocol requires only 0.3s connection setup time and 2MB transmission size for a 1K rule set with a 32-bit rule length. In addition, the filtering time for each packet is within 5μs. Compared with the traditional approach [15], the proposed DPF protocol can significantly reduce the communications overhead and the connection setup time, while both the user and rule privacy are guaranteed to be preserved.

The rest of the paper is organized as follows. In Section II, we formally formulate the privacy preserving problem of the deep packet filtering function. Section III introduces the current off-the-self cryptographic solution and explain the disadvantages in SDNs. Inspired by Section III, we propose a new DPF protocol in Section IV and formally prove the privacy preserving properties in Section V. Then, the communication overhead is discussed in Section VI. The implementation results and efficiency analysis over an HP SDN/NFV platform are presented in Section VII and VIII, respectively. Finally, we conclude this paper in Section IX.

II. Problem Formation

For DPF in SDNs, it is important for the network administrator to keep the filtering rules in private because the rules are usually valuable assets. On the other hand, the user is reluctant to reveal the data protected by the end-to-end security protocol due to the content privacy. To fulfill the privacy needs from both parties, we first define a private filtering function as follows.

Let $\mathcal{R} = \{\gamma_1, \gamma_2, ..., \gamma_n\}$ be a set of filtering rules for a specific field (e.g., a list of hostile URLs), where each $\gamma_i \in \{0,1\}^m$ represents an m-bit string. Let msg_t be the string of the field in t-th packet (e.g., the URL string). Let $\mathcal{F}$ denote the filtering function in a connection, where msg_t's are sent as follows.

$$\mathcal{F} = \{f_t\}_{t=1}^{T}$$

$$f_t(msg_t, \mathcal{R}) = \begin{cases} msg_t & \text{if } msg_t \in \mathcal{R} \\ \varnothing & \text{if } msg_t \notin \mathcal{R} \end{cases}$$

The user and the network securely compute $\mathcal{F}$ with the user's input msg_t's and the network owner's input $\mathcal{R}$. Afterwards, the network obtains the output of $\mathcal{F}$. Equipped with the above function, the network is not able to acquire the payload information unless it matches the filtering rules. For the privacy of the network owner, it is impossible for the user to acquire the rules from the network administrator.

In SDN environment, the controller that holds the rules needs to protect the input $\mathcal{R}$, while the VNFs that receives user's encrypted data are in charge of secure matching and filtering. It is necessary for a protocol to 1) securely realize $\mathcal{F}$, 2) diminish the overheads of the controller while protecting the rule set $\mathcal{R}$, and 3) achieve real-time matching and filtering in the VNFs. In the following, we first present a simple solution of DPF for SDNs.

III. Naive Cryptographic Solution

In this section, we introduce the 1-out-of-2 oblivious transfer protocol (OT) [14], a fundamental building block in secure computation. Then, we present a straightforward solution based on OT and explain its disadvantages in the SDN/NFV system.

The 1-out-of-2 OT is a two-party secure protocol, where a sender transfers two pieces of information, and a receiver obtains only one of them. The sender is not able to identify the received piece, and the receiver is also blind to the other piece. The naive approach exploits the following OT protocol to generate the encrypted rules and the encrypted data streams.

Definition 1: $\binom{2}{1}$-OT$_\ell^m$ denotes m times of the 1-out-of-2 oblivious transfer protocol over ℓ-bit strings. A sender and a receiver run the protocol with the following input and output.
Input: The sender's input is m pairs of (x_0^j, x_1^j) with $x_0^j, x_1^j \in \{0,1\}^\ell$, and the receiver's input is an m-bit vector $y = (y_1, y_2, ..., y_m)$ with $y_j \in \{0,1\}$.
Output: The receiver's output is $x_{y_j}^j$'s with $1 \le j \le m$.
During the protocol operation, the sender cannot learn the receiver's input, and the receiver only obtains the output.

A naive solution based on $\binom{2}{1}$-OT$_\ell^m$ that securely realizes the filtering function f_t for packet msg_t is explained as follows.

1) The user (i.e., the sender) and the SDN controller (i.e.,the receiver that executes $\binom{2}{1}$-OT$_\ell^m$ for each rule γ_i with $\gamma_i \in \{0,1\}^m$ and $1 \le i \le n$) operate as follows.
   a) The user randomly generates $n \times m$ pairs of (x_0^{ij}, x_1^{ij}) with $1 \le i \le n$ and $1 \le j \le m$ as the input of the sender.
   b) The SDN controller inputs $\gamma_i = b_{i1}b_{i2}...b_{im}$ with $b_{ij} \in \{0,1\}$ and then obtains $x_{b_{ij}}^{ij}$'s, the output of n times $\binom{2}{1}$-OT$_\ell^m$.
2) The SDN controller computes $\eta_i = x_{b_{i1}}^{i1} \oplus x_{b_{i2}}^{i2} \oplus ... \oplus x_{b_{im}}^{im}$ with $1 \le i \le n$ and installs all η_i's in the VNF.
3) Given $msg_t = c_1c_2...c_m$ with $c_j \in \{0,1\}$, the user computes $\delta_i = x_{c_1}^{i1} \oplus x_{c_2}^{i2} \oplus ... \oplus x_{c_m}^{im}$ with $1 \le i \le n$, and all δ_i's are then directed to the VNF for rule matching.
4) The VNF checks if $\delta_i = \eta_i$ holds. If it holds, the VNF learns that the t-th incoming packet violates the rule γ_i.

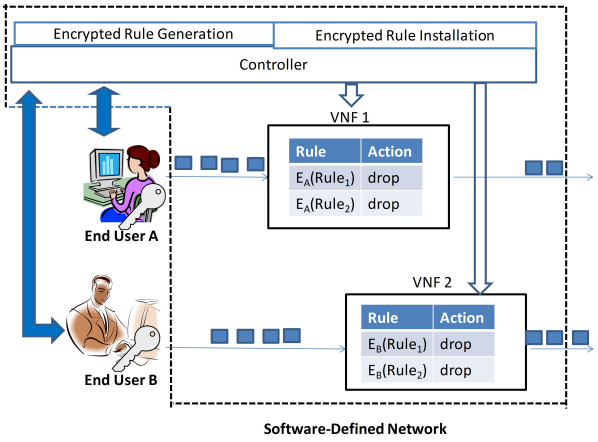

Fig. 2. The system model over the SDN/NFV platform

Otherwise, the network is not able to acquire the content of msg_t.

In Step 1) of the above protocol, it is worth noting that n times $\binom{2}{1}$-OT$_\ell^m$ for each packet is the bottleneck of the DPF service. The recent works [2][13] accelerate the OT protocol by exploiting the OT extension technique so that it only takes sub-seconds and a few megabytes transmission to execute thousands of $\binom{2}{1}$-OT$_\ell^m$. Nevertheless, it is expected that the above improvement is not sufficient to meet the requirement of DPF in large-scale SDNs. Moreover, the naive protocol is inclined to incur high overheads for the SDN controller because the controller is necessary to receive and install different η_i's when each packet forwarding occurs. Hence, in the following section, we design a new DPF protocol that effectively reduces the controller overheads and increases the filtering speed in VNFs at the same time, while both user privacy and rule privacy are guaranteed to be preserved.

IV. The Proposed Protocol

In this section, we describe the system model and the details of the proposed protocol.

A. System Model and Protocol Operation

Fig. 2 presents the system model. The network administrator manages the controller and VNFs, while each user intends to deliver a private data stream. The proposed protocol includes two phases: *setup phase* and *process phase*. The setup phase only needs to be performed once before the end-to-end secure channel is built, while and the VNFs then facilitates the filtering rules in the process phase.

Setup Phase. First, the SDN controller informs the user of the header fields that are required to be examined. Then, the user and the SDN controller run an interactive protocol to encrypt a set of the rules, where the user inputs the self-generated key, and the controller inputs a set of filtering rules. During the protocol operation, each party (i.e., the user and controller) is not able to learn the input of the other. At the end of the protocol, the SDN controller obtains the encrypted rules and installs them in the VNFs to perform filtering.

The Process Phase. First, for each forwarding packet, the user encrypts the fields with the same key, attaches the encrypted fields (called tokens) to the encrypted packet, and then sends the packet. After the packet is directed to a VNF, if the token attached in the encrypted packet matches the corresponding encrypted rules, the VNF drops the encrypted packet as specified in the rule. Otherwise, it keeps forwarding the packet.

In the following, we explain the details of the two phases.

B. The Setup Phase

Let $\mathcal{R} = \{\gamma_1, \gamma_2, ..., \gamma_n\}$ denote the rule set with n rules for the SDN controller, where each rule $\gamma_i \in \{0,1\}^m$ is an m-bit string. The user performs the following interactive protocol with the SDN controller as follows.

1) The user randomly generates a key set $\mathcal{K} = \{(k_{1,0}, k_{1,1}), (k_{2,0}, k_{2,1}), ..., (k_{m,0}, k_{m,1}), r\}$, where each pair of $(k_{j,0}, k_{j,1}) \in_R \{0,1\}^{2\ell}$ and $r \in_R \{0,1\}^\ell$ are random strings with ℓ bits.
2) For each rule γ_i, the user and the SDN controller executes the following steps.
   a) The user randomly generates a vector $(r_1^i, r_2^i, ..., r_m^i)$, where the vector satisfies $r = r_1^i \oplus r_2^i \oplus ... \oplus r_m^i$. Then the user computes the pairs $(\kappa_{j,0}^i, \kappa_{j,1}^i)$'s according to the following equations,
   $$\begin{matrix} \kappa_{j,0}^i = k_{j,0} \oplus r_j^i \\ \kappa_{j,1}^i = k_{j,1} \oplus r_j^i \end{matrix}, \text{ where } 1 \le i \le n \text{ and } 1 \le j \le m.$$
   b) The user and the SDN controller run a $\binom{2}{1}$-OT$_\ell^m$, where the user plays the sender with the input $(\kappa_{j,0}^i, \kappa_{j,1}^i)$'s, and the SDN controller acts as the receiver with the input γ_i. At the end, the SDN controller obtains $\kappa_{j,b_{ij}}^i$'s, where the rule $\gamma_i = b_{i1}b_{i2}....b_{im}$ represents a binary string with $b_{ij} \in \{0,1\}$.
   c) The SDN controller computes $\hat{\gamma}_i \in \{0,1\}^\ell$ as follows, where $\hat{\gamma}_i$ denotes the ciphertext of γ_i with the encryption key $\mathcal{K}$ denoted as $E_\mathcal{K}(\gamma_i)$.
   $$\begin{aligned} \hat{\gamma}_i &:= \kappa_{1,b_{i1}}^i \oplus \kappa_{2,b_{i2}}^i \oplus ... \oplus \kappa_{m,b_{im}}^i \\ &= k_{1,b_{i1}} \oplus k_{2,b_{i2}} \oplus ... \oplus k_{m,b_{im}} \oplus r \\ &= E_\mathcal{K}(\gamma_i) \end{aligned}$$
3) After n times repetition of the above steps, the SDN controller obtains a set of encoded rules $\hat{\mathcal{R}} = \{\hat{\gamma}_1, \hat{\gamma}_2, ..., \hat{\gamma}_n\}$ and installs $\hat{\mathcal{R}}$ in the designated VNF.

C. The Process Phase

Before sending out each encrypted packet, the user generates the token from the specific fields in the header of the packet for rule matching. Let $msg_t \in \{0,1\}^m$ denote the data of the matching fields in the t-th packet. The phase includes the procedures of token generation and rule matching, which are explained as follows.

1) The user computes $E_{\mathcal{K}/r}(msg_t)$ in the following equation, where $msg_t = c_1c_2...c_m$ for each $c_j \in \{0,1\}$.

$$E_{\mathcal{K}/r}(msg_t) = k_{1,c_1} \oplus k_{2,c_2} \oplus ... \oplus k_{m,c_m}$$

2) The user computes and attaches the token (α_t, β_t) to the t-th encrypted packet, where $\omega_t \in \{0,1\}^\ell$ denotes a random string and $H : \{0,1\}^\ell \rightarrow \{0,1\}^\lambda$ denotes a one-way cryptographic hash function.

$$\begin{aligned} \alpha_t &= H(E_{\mathcal{K}/r}(msg_t) \oplus \omega_t) \\ \beta_t &= r \oplus \omega_t \end{aligned}$$

3) The VNF then matches the token (α_t, β_t) with the corresponding rule set $\hat{\mathcal{R}} = \{\hat{\gamma}_1 \hat{\gamma}_2, ..., \hat{\gamma}_n\}$. If the equation $\alpha_t = H(\hat{\gamma}_i \oplus \beta_t)$ holds, the VNF learns $msg_t = \gamma_i$ and drops the t-th encrypted packet. Otherwise, the VNF keeps forwarding the packet.

V. Privacy Analysis

In this section, we first describe the attack model and then provide the formal privacy proof. More specifically, we prove that the privacy of the rules is successfully protected against the user in the setup phase, and the content of the packets remains unknown to the network if no rule is matched in the process phase.

A. Attack Model

In a two-party secure computation protocol, the attacker is one of the participants who attempts to obtain the information of the other's input. A participant who can learn more information than the designated output is regarded as a successful attacker. The ability of the attacker $\mathcal{A}$ is explained as follows. A passive and static attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is an entity who corrupts one of the participants at the beginning of the protocol and honestly follows the protocol. The subroutine $\mathcal{A}_1$ is a corrupted user who knows the traffic data. The other subroutine $\mathcal{A}_2$ is a corrupted network administrator who knows the rule set and acquires the output of the filtering function. Both subroutines can observe the transmitted data of the protocol, called the *view*, during the protocol operation.

B. Theoretic Proof

The goal of the privacy is to ensure that one (the user or the network administrator) does not leak any input information to the other. The strategy of the proof is to show that the information leakage of the proposed protocol is identical to that of the ideal one, which relies on a trusted third part. An attacker in the ideal one is called a simulator because its strategy is to simulate the view of the attacker in the real world. If the simulation is indistinguishable from the proposed protocol with respect to the attacker's view, it implies that the proposed protocol is as secure as the ideal one. The details of the proof are explained as follows.

The protocol withstanding the above attack model can ensure the user privacy and the rule privacy, increasing the incentive to deploy the DPF service. However, in practical situations, a stronger attacker may exist, such as an uncooperative user who deliberately dodges the filtering process. In this paper, we only focus on the privacy of the protocol under the curious-but-honest attacker and leave the above strong attackers in the future works.

Theorem 1: Based on the security of the oblivious transfer protocol and the cryptographic one-way hash function, the network administrator in the proposed protocol reveals no information of the rule set $\mathcal{R}$, and the user in the proposed protocol reveals no information of msg_t if $msg_t \notin \mathcal{R}$ under the static and passive attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

Proof: We divide the privacy proof into two parts: the rule privacy and the user privacy.

For the rule privacy, the user acts as the attacker $\mathcal{A}_1$ who attempts to learn the rule information. The view of $\mathcal{A}_1$ in the proposed protocol is the same as that of the sender in the $\binom{2}{1}$-OT$_\ell^m$ protocol. Since $\binom{2}{1}$-OT$_\ell^m$ has been formally proved, the user learns nothing as the sender in the oblivious transfer protocol.

For the user privacy, the network administrator is the attacker $\mathcal{A}_2$ who tries to learn the information of msg_t when $msg_t \notin \mathcal{R}$ occurs. The view of $\mathcal{A}_2$ is $(\kappa^i_{1,b_{i1}}, \kappa^i_{2,b_{i2}}, ..., \kappa^i_{m,b_{im}})$'s with each $\gamma_i = b_{i1}b_{i2}...b_{im}$ in the setup phase and (α_t, β_t)'s in the process phase, respectively. The simulator $\mathcal{S}$ in the ideal world acts as the user in the real protocol to simulate the network owner's following view, where the hash function is treated as a random oracle controlled by $\mathcal{S}$. In the setup phase, $\mathcal{S}$ generates the m pairs of random strings, $(\widetilde{\kappa}^i_{j,0}, \widetilde{\kappa}^i_{j,1}) \in_R \{0,1\}^{2\ell}$ for each $\gamma_i \in \mathcal{R}$ as the sender's input of $\binom{2}{1}$-OT$_\ell^m$. By following the protocol, $\mathcal{A}_2$ computes $\widetilde{\gamma}_i := \widetilde{\kappa}^i_{1,b_{i1}} \oplus \widetilde{\kappa}^i_{2,b_{i2}} \oplus ... \oplus \widetilde{\kappa}^i_{m,b_{im}}$. In the process phase, $\mathcal{S}$ randomly generates $\widetilde{\alpha}_t \in_R \{0,1\}^\lambda$ and $\widetilde{\beta}_t \in_R \{0,1\}^\ell$. After obtaining $(\widetilde{\alpha}_t, \widetilde{\beta}_t)$'s, the network owner will query the random oracle for the hash value of $\widetilde{\gamma}_i \oplus \widetilde{\beta}_t$. Then, the oracle responds $h^t_i \in_R \{0,1\}^\lambda / \widetilde{\alpha}_t$ to the query.

In the following, we prove that the view of $\mathcal{A}_2$ in the simulation is indistinguishable from that in the real protocol. In the setup phase, the receiving $\kappa^i_{j,b_{ij}}$'s that contains the random factor r^i_j's in the real protocol and $\widetilde{\kappa}^i_{j,b_{ij}}$'s in the simulation are both uniformly distributed over $\{0,1\}^\ell$. In the process phase, α_t's and β_t's in the real protocol are uniformly distributed over $\{0,1\}^\lambda$ and $\{0,1\}^\ell$ similar to $\widetilde{\alpha}_t$'s and $\widetilde{\beta}_t$'s. The reason is that (α_t, β_t) contains the random factor ω_t, and H is considered as a random oracle. For the matching result of $msg_t \notin \mathcal{R}$, the response h^t_i from the simulator ensures that the inequality $\widetilde{\alpha}_t \neq H(\widetilde{\gamma}_i \oplus \widetilde{\beta}_t)$ always holds. Therefore, the attacker $\mathcal{A}_2$ cannot distinguish the real protocol from the simulation. ■

VI. Discussion

The $\binom{2}{1}$-OT$_\ell^m$ extension requires $2m\ell$-bit transmissions for each party. In DPF-OT, each packet runs n times $\binom{2}{1}$-OT$_\ell^m$. Compare with DPF-OT, our proposed DPF-ET only needs to run once in the setup phase, while the user does not interact with the network in the process phase. By contrast, BlindBox runs AES-128 secure computation based on Yao's garbled circuit protocol in the setup phase that invokes the transmission of AES-128 garbled circuit and OT operation. An AES-128 circuit includes 9100 non-xor gates [12], and each non-xor gates requires 64 bytes to represent a garbled gate. Therefore, the size of an AES-128 garbled circuit is over 500KB. Compared to BlindBox, our DPF-ET does not incur

TABLE I
COMPARISON OF COMMUNICATIONS OVERHEAD

	The Setup Phase	The Process Phase (/pkt)
DPF-OT(User TX)	-	$2m \times \ell \times n$=1MB
DPF-OT(Network TX)	-	$2m \times \ell \times n$=1MB
BlindBox(User TX)	$(AES\_GC+2m\times \ell) \times n$=501MB	$\|RS\|$=5B
BlindBox(Network TX)	$2m \times \ell \times n$=1MB	-
DPF-ET(User TX)	$2m \times \ell \times n$=1MB	$\lambda + \ell$=24B
DPF-ET(Network TX)	$2m \times \ell \times \|\mathcal{R}\|$=1MB	-

[1] m=32 bits: the bit length of a rule
[2] ℓ=128 bits: the bit length of an OT string
[4] n=1000: the size of a rule table
[5] λ=64 bits: the output length of H
[6] AES_GC ≈500KB: the size of an AES-128 garbled circuit
[7] $|RS|$=40 bits: the modular length for computing a message digest

high communications overheads for a large rule set in the setup phase.

In the setup phase, a traffic sender prepares the keys for data encryption and runs the $\binom{2}{1}$-OT_ℓ^m extension to install the rules in VNFs via the SDN controller. DPF-ET only needs 2 MB in total for 1000 rules, but BlindBox requires 500 MB. Therefore, BlindBox incurs a much larger overhead on the SDN controller to interpret the messages. After the setup phase, a sender operates XOR on packet payloads with the keys generated in the setup phase and a random seed (β_t) generated for each single packet. Then, the sender hashes the payloads and inserts (α_t, β_t) into the beginning of the payloads before sending the packet.

Compared to the DPF-OT (introduced in Section III), the forwarding procedure of the DPF-EF is much faster, because the the DPF-OT needs to finish OT operations on all rules for each packet, and each OT operation requires 2 MB transmission in total. Therefore, 2 GB messages are required for a packet when the size of the rule table is 1000. In other words, the OT operations are expensive and incur high overheads on the SDN controller. Furthermore, the forwarding procedure of DPF-EF is also faster than Blindbox, because Blindbox needs to encrypt all rules with AES, which is slower than XOR.

VII. IMPLEMENTATION

We implement the proposed protocol in an OpenFlow experimental network with five HP servers (HP DL320e Gen8) and five HP OpenFlow switches (three HP 5406zl and two HP 3800) to evaluate the performance of DPF-ET. One of the servers acts as the traffic sender, while another server is configured as the SDN controller. In addition, we implement the proposed DPF-ET as virtualized network functions (VNFs) running in the other servers attached to SDN switches, which are managed by a service-chain platform (SCP) [22]. Packets are first forwarded to an attached VNF and sent back to the SDN switch if the packets are not dropped by DFP rules as shown in Figure 1. In each VNF, we run Click Software Router [21] to properly manage the traffic. To communicate with the SDN controller, we add a communication module in Click Software Router to open a socket connecting to the controller and receive the matching rules from the controller.

For the protocol implementation, we adopt the VMAC64 [24] in the Crypto++ library [26] as the hash function with the output length as $\lambda = 64$ bits. VMAC64 requires an additional key as the input, so we assume that the user and the network share a secret key in the setup phase. We also compare the proposed protocol with the open-source implementation [25] of the OT [10] and OT extension protocols [7], where $\ell = 128$ and $m = 32$ for the OT operation in the experiment.

In our implementation, the test traffic is generated randomly, and the traffic sender encrypts the payloads and inserts the tokens, i.e., 64-bit α_t's and 128-bit β_t's, in the head of the packet payloads. DPF VNFs are deployed and attached to the OpenFlow switches. After receiving the packets, DPF-EF fetches (α_t, β_t)'s from the beginning of the payloads and then scans the rule table. For each rule in the rule table, DPF-EF hashes and operates XOR on the rule with β_t and compares the rule with α_t. If one of the rules matches α_t, DPF-EF drops the packet. Otherwise, it passes the packet back to the OpenFlow switches. Both the traffic sender and VNF with DPF-ET are deployed in HP servers with Intel Xeon CPU E3-1230 3.3 GHz and 4 GB memory.

VIII. EVALUATION

In the following, we evaluate the proposed protocol with the following performance metrics. (1)The speed of packet matching. How fast can our solution match packets? (2) The overhead of different components. We measure the time spent on different parts of our solution. (3) The speed for packet encryption. How fast can a sender encrypt and send packets? We compare the proposed protocol with the traditional DPF without traffic encryption and the DPF with OT encryption (DPF-OT) introduced in Section III, which is also implemented in VNFs.

A. Payload Matching

In DPF-ET, we first XOR and hash all entries in the rule table and then match each packet payload with the entries. Based on the matching result, the packet will be dropped or forwarded. In this section, we analyze the overhead of packet encryption and the time for DPF in VNFs. Fig. 3 shows the matching speed in terms of packets per second with different table sizes. When the table size increases, it is necessary for DPF-ET to spend more time to properly handle the table entries. Nevertheless, the speed of our DPF is able to process 15k to 114k packets per second, while DPF-OT can serve only 92 to 823 packets per second. The traditional DPF can reach 52k to 287k packets per second, but the user privacy is violated. Therefore, if the packet length is 1400 bytes (i.e., the common packet length in the Internet), DPF-ET is able to handle the traffic as fast as 1.2 Gbps when the table size is 1000.

B. Overhead Analysis at the Network Side

To identify the bottleneck of DPF-ET, we divided DPF-ET into four parts: XOR, hash, matching, and packet handling. Packet handling includes reading packets from NIC, payload

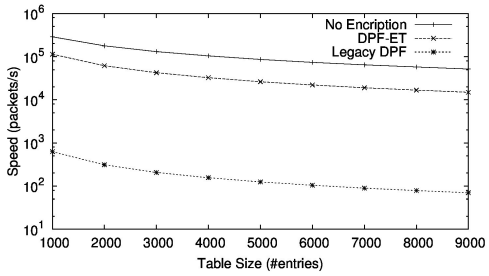

Fig. 3. The performance with different table sizes.

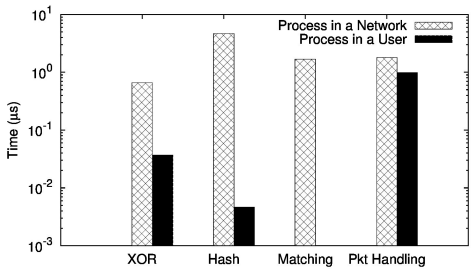

Fig. 4. The overhead of different components in DPF-ET

interpretation, and sending them back to the NIC. When a packet comes in, DPF-ET first interprets and fetches its payload and then scans the rule table. For each table entry, the rule is first handled by XOR and hash functions, and then the packet payload is compared with the rule. Figure 4 presents the time for each operation spent on a packet when the size of the rule table is 1000. The result manifests that the overhead in the network is higher than the overhead in the user. For the network side, the rule table is necessary to be scanned for rule matching, and both XOR and hash run 1000 times for each packet according to the process phase in DPF-ET. XOR spends less time than the others, but hash requires more time.

C. Overhead Analysis at the User Side

In the following, we evaluate the speed to encrypt the packets on the sender's side. Figure 4 shows the time in μs for each operation on each packet, and all operations are less then 1 μs. Therefore, the result manifests that the sender spends most time on packet processing, and both XOR and hash time can be ignored. Moreover, when the packet length is 1400-byte, the sending rate can achieve 10 Gbps.

D. Overhead Analysis in the Setup Phase

In the setup phase, the SDN controller performs rule encryption with the client before installing the encrypted rules in VNFs. To encrypt the rules, it is necessary to finish OT operations in Step 2 of Section IV, together with some extra XOR operations in DPF-ET. TABLE II shows the time required to install 1000 rules. The result manifests that OT operations take much longer time, and DPF-ET only incurs limited overheads with the overall setup time around 300 ms. Thus, DPF-ET is very efficient for DPF in SDNs.

TABLE II
PROCESS TIME IN THE SETUP PHASE.

	Time
OT	200 ms
Other OPs	1.232 ms

TABLE III
SETUP TIME COMPARISON.

	Time
DPF-ET	201.2 ms
BlindBox	9500 ms

Compared with BlindBox, DPF-ET requires much less setup time as shown in TABLE III. BlindBox requires 9.5 seconds, while DPF-ET only needs 201.2 ms. The high setup time in BlindBox is induced from the high volume of communications (502 MB V.S. 2 MB) and computation. By contrast, DPF is only involved in Setup Phase for each private connection, and DPF-ET thereby has much lower setup overheads and is able to support much more connections in SDNs.

IX. CONCLUSION

In this paper, we proposed DPF-ET, a privacy-preserving deep packet filtering protocol that allows the network owner performing efficient rule filtering over encrypted traffic data in SDN/NFV environment. With regard to privacy, the network owner does not able to learn the packet data, while the users cannot acquire the filtering rules. The implementation results on an HP SDN/NFV system manifest that DPF-ET only invokes 2MB communication overhead in the setup phase and takes 5 μs per packet in the process phase for matching a 1K rule set with a 32-bit rule length.

REFERENCES

[1] R. Alshammari and A. N. Zincir-Heywood, "Machine Learning based encrypted traffic classification: identifying SSH and Skype," IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1-8, 2009.

[2] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," In Proceedings of the ACM SIGSAC conference on Computer and Communications Security, pp. 535-548, 2013.

[3] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," In Proceedings of the 1st ACM conference on Computer and communications security, pp. 62-73, ACM Press, 1993.

[4] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai, "Universally Composable Two-Party and Multi-Party Secure Computation," In Proceedings of ACM Symposium on Theory of Computing (pp. 494-503), 2002.

[5] A. Dainotti, A. Pescape and K. C. Claffy, "Issues and future directions in traffic classification," IEEE Network, 26(1),pp. 35-40, 2012.

[6] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: flexible and elastic DDoS defense," 24th USENIX Security, pp.817-832, 2015.

[7] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," In Advances in Cryptology-CRYPTO03, Vol. 2729 of LNCS, pp. 145-161. Springer, 2003.

[8] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review 38(2), pp. 69-74, 2008.

[9] S. A. Mehdi, J. Khalid, and S. A. Khayam, Revisiting traffic anomaly detection using software defined networking, in Proc. 14th Int. Symp. Recent Advances in Intrusion Detection (RAID), 2011, vol. 6961, pp. 161V180.

[10] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2001.

[11] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," Communications Surveys & Tutorials, IEEE, 10(4), pp. 56-76, 208.
[12] B. Pinkas, T. Schneider, N. P. Smart, S. C. Williams, "Secure two-party computation is practical," In Advances in CryptologyVASIACRYPT, pp. 250-267, 2009.
[13] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," Usenix Security, Vol. 14, pp. 797-812, 2014.
[14] M. O. Rabin, "How to exchange secrets with oblivious transfer," TR-81 edition, 1981, Aiken Computation Lab, Harvard University.
[15] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep Packet Inspection over Encrypted Traffic," SIGCOMM Conference, 2015.
[16] A. C. Yao, "How to Generate and Exchange Secrets," In Proc. IEEE FOCS, 1986.
[17] Exfiltration Prevention, http://www.filetransferconsulting.com/exfiltration-definition-security-risk-prevention/
[18] talktalk HomeSafe, http://www.talktalk.co.uk/security/faq/
[19] SSL Visibility and Management, https://www.bluecoat.com/products/ssl-decryption-visibility-and-management
[20] Network Functions Virtualisation, https://portal.etsi.org/NFV/NFV_White_Paper.pdf
[21] Click Software Router, http://read.cs.ucla.edu/click/click
[22] Service-Chain Platform, http://islab.iis.sinica.edu.tw/dokuwiki/
[23] Hash Algorithm Benchmark, http://www.cryptopp.com/benchmarks-amd64.html
[24] VMAC website, http://www.fastcrypto.org/vmac/
[25] OT extension implementation https://github.com/encryptogroup/OTExtension
[26] Crypto++ library https://www.cryptopp.com/