Pine: Enabling Privacy-Preserving Deep Packet Inspection on TLS with Rule-Hiding and Fast Connection Establishment

Jianting Ning[1,4], Xinyi Huang[1(✉)], Geong Sen Poh[2], Shengmin Xu[1], Jia-Chng Loh[2], Jian Weng[3], and Robert H. Deng[4]

[1] Fujian Provincial Key Laboratory of Network Security and Cryptology, College of Mathematics and Informatics, Fujian Normal University, Fuzhou, China
jtning88@gmail.com, xyhuang81@gmail.com, smxu1989@gmail.com
[2] NUS-Singtel Cyber Security Lab, Singapore, Singapore
pohgs@comp.nus.edu.sg, dcsljc@nus.edu.sg
[3] College of Information Science and Technology, Jinan University, Guangzhou, China
cryptjweng@gmail.com
[4] School of Information Systems, Singapore Management University, Singapore, Singapore
robertdeng@smu.edu.sg

Abstract. Transport Layer Security Inspection (TLSI) enables enterprises to decrypt, inspect and then re-encrypt users' traffic before it is routed to the destination. This breaks the end-to-end security guarantee of the TLS specification and implementation. It also raises privacy concerns since users' traffic is now known by the enterprises, and third-party middlebox providers providing the inspection services may additionally learn the inspection or attack rules, policies of the enterprises. Two recent works, BlindBox (SIGCOMM 2015) and PrivDPI (CCS 2019) propose privacy-preserving approaches that inspect encrypted traffic directly to address the privacy concern of users' traffic. However, BlindBox incurs high preprocessing overhead during TLS connection establishment, and while PrivDPI reduces the overhead substantially, it is still notable compared to that of TLSI. Furthermore, the underlying assumption in both approaches is that the middlebox knows the rule sets. Nevertheless, with the services increasingly migrating to third-party cloud-based setting, rule privacy should be preserved. Also, both approaches are static in nature in the sense that addition of any rules requires significant amount of preprocessing and re-instantiation of the protocols.

In this paper we propose Pine, a new Privacy-preserving inspection of encrypted traffic protocol that (1) simplifies the preprocessing step of PrivDPI thus further reduces the computation time and communication overhead of establishing the TLS connection between a user and a server; (2) supports *rule hiding*; and (3) enables dynamic rule addition without the need to re-execute the protocol from scratch. We demonstrate the

L. Chen et al. (Eds.): ESORICS 2020, LNCS 12308, pp. 3–22, 2020.
https://doi.org/10.1007/978-3-030-58951-6_1

superior performance of Pine when compared to PrivDPI through extensive experimentations. In particular, for a connection from a client to a server with 5,000 tokens and 6,000 rules, Pine is approximately 27% faster and saves approximately 92.3% communication cost.

Keywords: Network privacy · Traffic inspection · Encrypted traffic

1 Introduction

According to the recent Internet trends report [11], 87% of today's web traffic was encrypted, compared to 53% in 2016. Similarly, over 94% of web traffic across Google uses HTTPS encryption [7]. The increasing use of end-to-end encryption to secure web traffic has hampered the ability of existing middleboxes to detect malicious packets via deep packet inspection on the traffic. As a result, security service providers and enterprises deploy tools that perform Man-in-the-Middle (MitM) to decrypt, inspect and re-encrypt traffic before the traffic is sent to the designated server. Such approach is termed as Transport Layer Security Inspection (TLSI) by the National Security Agency (NSA), which recently issued an advisory on TLSI [12] citing potential security issues including insider threats. TLSI introduces additional risks whereby administrators may abuse their authorities to obtain sensitive information from the decrypted traffic. On the other hand, there exists growing privacy concern on the access to users' data by middleboxes as well as the enterprise gateways. According to a recent survey on TLSI in the US [16], more than 70% of the participants are concerned that middleboxes (or TLS proxies) performing TLSI can be exploited by hackers or used by governments, and close to 50% think it is an invasion to privacy. In general, participants are acceptable to the use of middleboxes by their employers or universities for security purposes but also want assurance that these would not be used by governments for surveillance or by exploited hackers.

To alleviate the above concerns on maintaining security of TLS while ensuring privacy of the encrypted traffic, Sherry *et al.* [20] introduced a solution called BlindBox to perform inspection on encrypted traffic directly. However, BlindBox needs a setup phase that is executed between the middlebox and the client. The setup phase performs two-party computation where the input of the middlebox are the rules, which means that the privacy of rules against the middlebox is not assured. In addition, this setup phase is built based on garbled circuit, and needs to be executed for every session. Due to the properties of garble circuit, such setup phase incurs significant computation and communication overheads. To overcome this limitation, Ning *et al.* [15] recently proposed PrivDPI with an improved setup phase. A new obfuscated rule generation technique was introduced, which enables the reuse of intermediate values generated during the first TLS session across subsequent sessions. This greatly reduces the computation and communication overheads over a series of sessions. However, there still exists considerable delay during the establishment of a TLS connection since each client is required to run a preprocessing protocol for each new connection. In addition,

as we will show in Sect. 4.1, when the domain of the inspection or attack rules is small, the middlebox could perform brute force guessing for the rules in the setting of PrivDPI. This means that, as in BlindBox, PrivDPI does not provide privacy of rules against the middlebox. However, as noted in [20], most solution providers, such as McAfee, rely on the privacy of their rules in their business model. More so given the increasingly popular cloud-based middlebox services, the privacy of the rules should be preserved against the middleboxes.

Given the security and privacy concerns on TLSI, and the current status of the state-of-the-arts, we seek to introduce a new solution that addresses the following issues, in addition to maintaining the security and privacy provisions of BlindBox and PrivDPI: *(1) Fast TLS connection establishment without preprocessing in order to eliminate the session setup delay incurred in both BlindBox and PrivDPI; (2) Resisting brute force guessing of the rule sets even for small rule domains; (3) Supporting lightweight rule addition.*

Our Contributions. We propose Pine, a new protocol for privacy-preserving deep packet inspection on encrypted traffic, for a practical enterprise network setting, where clients connect to the Internet through an enterprise gateway. The main contributions are summarized as follows.

- **Identifying limitation of PrivDPI.** We revisit PrivDPI and demonstrate that in PrivDPI, when the rule domain is small, the middlebox could forge new encrypted rules that gives the middlebox the ability to detect the encrypted traffic with any encrypted rules it generates.
- **New solution with stronger privacy guarantee.** We propose Pine as the new solution for the problem of privacy-preserving deep packet inspection, where stronger privacy is guaranteed. First of all, the privacy of the traffic is protected unless there exists an attack in the traffic. Furthermore, privacy of rules is assured against the middlebox, we call this property *rule hiding*. This property ensures privacy of rules even when the rule domain is small (e.g. approximately 3000 rules as in existing Network Intrusion Detection (IDS) rules), which addresses the limitation of PrivDPI. In addition, privacy of rules is also assured against the enterprise gateway and the endpoints, we term this property *rule privacy*.
- **Amortized setup, fast connection establishment.** Pine enables the establishment of a TLS connection with low latency and without the need for an interactive preprocessing protocol as in PrivDPI and BlindBox. The latency-incurring preprocessing protocol is performed offline and is only executed once. Consequently, there is no per-user-connection overhead. Any client can setup a secure TLS connection with a remote server without preprocessing delay. In contrast, in PrivDPI and BlindBox, the more rules there are, the higher the per-connection setup cost is. The speed up of the connection is crucial for low-latency applications.
- **Lightweight rule addition**. Pine is a dynamic protocol in that it allows new rules being added on the fly without affecting the connection between a client and a server. The rule addition is seamless to the clients in the sense that the gateway can locally execute the rule addition phase with the middlebox

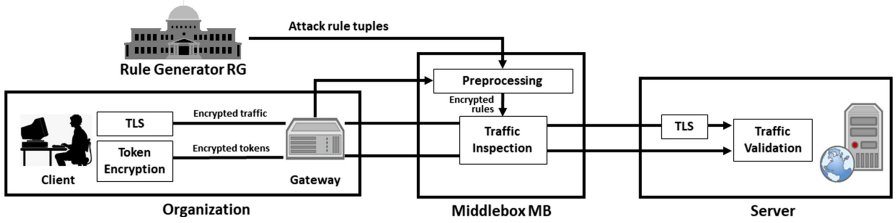

Fig. 1. Pine system architecture.

without any client involvement. This is beneficial as compared to BlindBox and PrivDPI, where the client would need to re-run the preprocessing protocol from scratch for every connection.

In addition to stronger privacy protection, we conduct extensive experiments to demonstrate the superior performance of Pine when compared to PrivDPI. For a connection from a client to a server with 5,000 tokens and a ruleset of 6,000, Pine is approximately 27% faster than PrivDPI, and saves approximately 92.3% communication cost. In particular, the communication cost of Pine is independent of the number of rules, while the communication cost of PrivDPI grows linear with the number of rules.

2 Protocol Overview

Pine shares a similar architecture with BlindBox and PrivDPI, as illustrated in Fig. 1. There are five entities in Pine: Client, Server, Gateway (**GW**), Rule Generator (**RG**) and Middlebox (**MB**). Client and server are the endpoints that send and receive network traffic protected by TLS. **GW** is a device located between a set of clients and servers that allows network traffic to flow from one endpoint to another endpoint. **RG** generates the attack rule tuples for **MB**. The attack rule tuples will be used by **MB** to detect attacks in the network traffic. Each attack rule describes an attack and contains one or more keywords to be matched in the network communication. Hereafter, we will use the terms "rule" and "attack rule" interchangeably. The role of **RG** can be performed by organization such as McAfee [18]. **MB** is a network device that inspects and filters network traffic using the attack rule tuples issued by **RG**.

System Requirements. The primary aim is to provide a privacy-preserving mechanism that can detect any suspicious traffic while at the same time ensure the privacy of endpoint's traffic. In particular, the system requirements include:

- *Traffic inspection*: Pine retains similar functionality of traditional IDS, i.e., to find a suspicious keyword in the packet.
- *Rule privacy*: The endpoints and **GW** should not learn the attack rules (i.e., the keywords). This is required especially for security solution providers that generate comprehensive and proprietary rule sets as their unique proposition that help to detect malicious traffic more effectively.

- *Traffic privacy*: On one hand, **MB** is not supposed to learn the plaintexts of the network traffic, except for the portions of the traffic that match the rules. On the other hand, **GW** is not allowed to read the content of the traffic.
- *Rule hiding*: **MB** is not supposed to learn the attack rules from the attack rule tuples issued by **RG** in a cloud-based setting where **MB** resides on a cloud platform. In such a case the cloud-based middlebox is not fully trusted. The security solution providers would want to protect the privacy of their unique rule sets, as was discussed previously in describing rule privacy.

Threat Model. There are three types of attackers described as follows.

- *Malicious endpoint.* The first type of attacker is the endpoint (i.e., the client or the server). Similar to BlindBox [20] and PrivDPI [15], at most one of the two endpoints is assumed to be malicious but not both. Such an attacker is the same as the attacker in the traditional IDS whose main goal is to evade detection. As in the traditional IDS [17], it is a fundamental requirement that at least one of the two endpoints is honest. This is because if two malicious endpoints agree on a private key and send the traffic encrypted by this particular key, detection of malicious traffic would be infeasible.
- *The attacker at the gateway.* As in conventional network setting, **GW** is assumed to be semi-honest. That is, **GW** honestly follows the protocol specification but may try to learn the plaintexts of the traffic. **GW** may also try to infer the rules from the messages it received.
- *The attacker at the middlebox.* **MB** is assumed to be semi-honest, which follows the protocol but may attempt to learn more than allowed from the messages it received. In particular, it may try to read the content of the traffic that passed through it. In addition, it may try to learn the underlying rules of the attack rule tuples issued by **RG**.

Protocol Flow. We present how each phase functions at a high level as follows.

- Initialization. **RG** initializes the system by setting the public parameters.
- Setup. **GW** subscribes the inspection service from **RG**, in which **RG** receives a shared secret from **GW**. **RG** issues the attack rule tuples to **MB**. The client and the server will derive some parameters from the key of the primary TLS handshake protocol and install a Pine HTTPS configuration, respectively.
- Preprocessing. In this phase, **GW** interacts with **MB** to generate a set of reusable randomized rules. In addition, **GW** generates and sends the initialization parameters to the clients within its domain.
- Preparation of Session Detection Rule. In this phase, the reusable randomized rules will be used to generate session detection rules.
- Token Encryption. In this phase, a client generates the encrypted token for each token in the payload. The encrypted tokens will be sent along with the traffic encrypted from the payload using regular TLS.
- Gateway Checking. For the first session, **GW** checks whether the attached parameters sent by the client is well-formed. This phase will be run when a client connects to a server for the first time.

- Traffic Inspection. **MB** generates a set of encrypted rules and performs inspection using these encrypted rules.
- Traffic Validation. One endpoint performs traffic validation in case the other endpoint is malicious.
- Rule Addition. A set of new attack rules will be added in this phase. **GW** interacts with **MB** to generate the reusable randomized rule set corresponding to these new attack rules.

3 Preliminaries

Complexity Assumption. The decision Diffie-Hellman (DDH) problem is stated as follows: given g, g^x, g^y, g^z, decide whether $z = xy$ (modulo the order of g), where $x, y, z \in \mathbb{Z}_p$. We say that a PPT algorithm $\mathcal{B}$ has advantage ϵ in solving the DDH problem if $|\Pr[\mathcal{B}^{(g,g^x,g^y,g^{xy})} = 1] - \Pr[\mathcal{B}^{(g,g^x,g^y,g^z)} = 1]| \geq \epsilon$, where the probability above is taken over the coins of $\mathcal{B}$, g, x, y, z.

Definition 1. *The DDH assumption holds if no PPT adversary has advantage at least ϵ in solving the DDH problem.*

Pseudorandom function. A pseudorandom function family PRF is a family of functions $\{\mathsf{PRF}_a : U \to V | a \in \mathcal{A}\}$ such that $\mathcal{A}$ could be efficiently samplable and all PRF, U, V, $\mathcal{A}$ are indexed by a security parameter λ. The security property of a PRF is: for any PPT algorithm $\mathcal{B}$ running in λ, it holds that $|\Pr[\mathcal{B}^{\mathsf{PRF}_a(\cdot)} = 1] - \Pr[\mathcal{B}^{R(\cdot)} = 1]| = \mathtt{negl}(\lambda)$, where $\mathtt{negl}$ is a negligible function of λ, a and R are uniform over $\mathcal{A}$ and $(U \to V)$ respectively. The probability above is taken over the coins of $\mathcal{B}$, a and R. For notational simplicity, we consider one version of the general pseudorandom function notion that is custom-made to fit our implementation. Specifically, the pseudorandom function PRF considered in this paper maps λ-bit strings to elements of $\mathbb{Z}_p$. Namely, $\mathsf{PRF}_a : \{0,1\}^\lambda \to \mathbb{Z}_p$, where $a \in G$.

Payload Tokenization. As in BlindBox and PrivDPI, we deploy window-based tokenization to tokenize keywords of a client's payload. Window-based tokenization follows a simple sliding window algorithm. We adopt 8 bytes per token when we implement the protocol. That is, given a payload "secret key", an endpoint will generate the tokens "secret k", "ecret ke" and "cret key".

4 Protocol

In this section, we first point out the limitation of PrivDPI. To address this problem and further reduce the connection delay, we then present our new protocol.

4.1 Limitation of PrivDPI

We show how PrivDPI fails when the domain of rule is small. We say that the domain of rule is small if one can launch brute force attack to guess the underlying rules given the public parameters. We first recall the setup phase of

PrivDPI. In the setup phase, a middlebox receives $(s_i, R_i, \mathsf{sig}(R_i))$ for rule r_i, where $R_i = g^{\alpha r_i + s_i}$ and $\mathsf{sig}(R_i)$ is the signature of R_i. With s_i and R_i, **MB** obtains the value $g^{\alpha r_i}$. Recall that in PrivDPI, the value $A = g^{\alpha}$ is included in the PrivDPI HTTPS configuration, **MB** could obtain this value via installing a PrivDPI HTTPS configuration. Since the domain of rule is small, with A and $g^{\alpha r_i}$, **MB** can launch brute force attack to obtain the value of r_i via trying every candidate value v by checking $A^v \stackrel{?}{=} g^{\alpha r_i}$ within the rule domain. In this way, **MB** could obtain the value r_i for R_i and r_j for R_j. After the completion of preprocessing protocol, **MB** obtains the reusable obfuscated rule $I_i = g^{k\alpha r_i + k^2}$ for rule r_i. Now, **MB** knows values r_i, r_j, $I_i = g^{k\alpha r_i + k^2}$, $I_j = g^{k\alpha r_j + k^2}$. It can then computes $(I_i/I_j)^{(r_i - r_j)^{-1}}$ to obtain a value $g^{k\alpha}$. With $g^{k\alpha}$, r_i and $I_i = g^{k\alpha r_i + k^2}$, it can compute $I_i/(g^{k\alpha})^{r_i} = g^{k^2}$. With g^{k^2} and $g^{k\alpha}$, **MB** could forge the reusable obfuscated rule successfully for any rule it chooses. With the forged (but valid) reusable obfuscated rule, **MB** could detect more than it is allowed, which violates the privacy requirement of the encrypted traffic.

4.2 Description of Our Protocol

Initialization. Let $\mathcal{R}$ be the domain of rules, PRF be a pseudorandom function, n be the number of rules and $[n]$ be the set $\{1, ..., n\}$. Let $\mathsf{AES}_a(\mathsf{salt})$ be the AES encryption with key a and message salt. Let $\mathsf{Enc}_a(\mathsf{salt}) = \mathsf{AES}_a(\mathsf{salt}) \mod \mathsf{R}$, where R is an integer used to reduce the ciphertext size [20]. The initialization phase takes in a security parameter λ and chooses a group G of prime order p. It then chooses a generator g of G, and sets the public parameters as (G, p, g).

Setup. GW chooses a key g^w for the pseudorandom function PRF, where $w \in \mathbb{Z}_p^*$. It subscribes the service from **RG** and sends w to **RG**. **RG** first computes $W = g^w$. For a rule set $\{r_i \in \mathcal{R}\}_{i \in [n]}$, for $i \in [n]$, **RG** chooses a randomness $k_i \in \mathbb{Z}_p$, calculates $r_{w,i} = \mathsf{PRF}_W(r_i)$ and $R_i = g^{r_{w,i} + k_i}$. **RG** chooses a signature scheme with sk as the secret key and pk as the public key. It then signs $\{R_i\}_{i \in [n]}$ with sk and generates the signature of R_i for $i \in [n]$, denote by σ_i. Finally, it sends the *attack rule tuples* $\{(R_i, \sigma_i, k_i)\}_{i \in [n]}$ to **MB**. Here, g^w is the key ingredient for ensuring the property of rule hiding. The key observation here is that since **MB** does not know g^w or w, it cannot guess the underlying r_i of R_i via brute forcing all the possible keywords it chooses. In particular, for a given attack rule tuple (R_i, σ_i, k_i), **MB** could obtain the value $g^{r_{w,i}}$ by computing R_i/g^{k_i}. Due to the property of pseudorandom function, $r_{w,i}$ is pseudorandom, and hence $g^{r_{w,i}}$ is pseudorandom. Without the knowledge of g^w or w, it is impossible to obtain r_i even if **MB** brute forces all possible keywords it chooses.

On the other hand, the client and the server install a Pine HTTPS configuration which contains a value R. Let k_{sk} be the key of the regular TLS handshake protocol established by a client and a server. With k_{sk}, the client (resp. the server) derives three keys k_T, c, k_s. Specifically, k_T is a standard TLS key, which is used to encrypt the traffic; c is a random value from $\mathbb{Z}_p$, which is used for generating session detection rules; k_s is a random value from $\mathbb{Z}_p$, which is used as a randomness to mask the parameters sent from the client to the server.

Preprocessing. In order to accelerate the network connection between a client and a server (compared to PrivDPI), we introduce a new approach that enables fast connection establishment without executing the preprocessing process per client as in PrivDPI. We start from the common networking scenario in an enterprise setting where there exists a gateway located between a set of clients and a server. The main idea is to let the gateway be the representative of the clients within its domain, who will run the preprocessing protocol with **MB** for only once. Both the clients and the gateway share the initialization parameters required for connection with the server. In this case, the connection between a client and a server can be established instantly without needing any preprocessing as in PrivDPI since the preprocessing is performed by the gateway and **MB** beforehand. In other words, we offload the operation of preprocessing to the gateway, which dramatically reduces the computation and communication overhead for the connection between a client and a server.

Specifically, in this phase, **GW** runs a preprocessing protocol with **MB** to generate a *reusable randomized rule* set as well as the initialization parameters for the clients within the domain of **GW**. The preprocessing protocol is run after the TLS handshake protocol, which is described in Fig. 2. Upon the completion of this phase, **MB** obtains a set of reusable randomized rules which enable **MB** to perform deep packet detection over the encrypted traffic across a series of sessions. The values I_0, I_1 and I_2 enable each client within the domain of **GW** to generate the encrypted tokens. Hence, for any network connection with a server, a client does not need to run the preprocessing phase with **MB** as compared to BlindBox and PrivDPI. This substantially reduces the delay and communication cost for the network connection between the client and the server, especially for large rule set. Furthermore, in case of adding new rules, a client does not need to re-run the preprocessing protocol as BlindBox and PrivDPI does. This means rule addition has no effect on the client side.

Preparation of Session Detection Rule. A set of session detection rules will be generated in this phase. These session detection rules are computed, tailored for every session, from the reusable randomized rules generated from the preprocessing protocol. The generated session detection rules are used as the inputs to generate the corresponding encrypted rules. The protocol is described in Fig. 3, and it is executed for every new session.

Token Encryption. Similar to BlindBox and PrivDPI, we adopt the window-based tokenization approach as described in Sect. 3. After the tokenization step, a client obtains a set of tokens corresponding to the payload. For the first time that a client connects with a server, the client derives a salt from c and stores the salt for future use, where c is the key derived from the key k_{sk} of the TLS handshake protocol. For each token t, a client runs the token encryption algorithm as described in Fig. 4. To prevent the count table T from growing too large, the client will clear T every Z sessions (e.g., $Z = 1,000$). In this case, the client will send a new salt to **MB**, where $\mathsf{salt} \leftarrow \mathsf{salt} + max_t \mathsf{count}_t + 1$.

In the above, we describe the token encryption when the endpoint is a client. When the endpoint is a server, the server will first run the same tokenization step, and encrypts the tokens as the step 1 and step 2 described in Fig. 4.

Gateway Checking. This phase will be executed when a client connects to a server for the first time. For the traffic sent from the client to a server for the first time, the client attaches $(\mathsf{salt}, C_{ks}, C_w, C_x, C_y)$. This enables the server to perform the validation of the encrypted traffic during the traffic validation phase. C_{ks} and k_s serve as the randomness to mask the values g^w, g^x and g^{xy}. The correctness of C_{ks} will be checked once the traffic reached the server. To ensure that g^w, g^x and g^{xy} are masked by C_{ks} correctly, **GW** simply checks whether the following equations hold: $C_w = (C_{ks})^w$, $C_x = (C_{ks})^x$ and $C_y = (C_{ks})^{xy}$.

Traffic Detection. During the traffic detection phase, **MB** performs the equality check between the encrypted tokens in the traffic and the encrypted rules it kept. The traffic detection algorithm is described as follows. **MB** first initializes a counter table CT_r to record the encrypted rule E_{r_i} for each rule r_i. The encrypted rule E_{r_i} for rule r_i is computed as $E_{r_i} = \mathsf{Enc}_{S_i}(\mathsf{salt} + \mathsf{count}_{r_i})$, where count_{r_i} is initialized to be 0. **MB** then generates a search tree that contains the encrypted rules. If a match is found, **MB** takes the corresponding action, deletes the old E_{r_i} corresponding to r_i, increases count_{r_i} by 1, computes and inserts a new E_{r_i} into the tree, where the new E_{r_i} is computes as $\mathsf{Enc}_{S_i}(\mathsf{salt} + \mathsf{count}_{r_i})$.

Traffic Validation. If it is the first session between a client and a server, upon receiving $(\mathsf{salt}, C_{ks}, C_w, C_x, C_y)$, the server checks whether the equation $C_{ks} = g^{k_s}$ holds, where k_s is derived (by the server) from the key k_{sk} of the regular TLS handshake protocol. If the equation holds, the server computes $(C_w)^{(k_s)^{-1}} = g^w$, $(C_x)^{(k_s)^{-1}} = g^x$, $(C_y)^{(k_s)^{-1}} = g^{xy}$. With the computed (g^w, g^x, g^{xy}), the server runs the same token encryption algorithm on the plaintext decrypted from the

Input: **MB** has inputs $\{(R_i, \sigma_i, k_i)\}_{i\in[n]}$, where $R_i = g^{r_{w,i}+k_i}$; **GW** has input pk.

The protocol is run between **GW** and **MB**:

1. **GW** chooses a random $x \in \mathbb{Z}_p^*$, computes $X = g^x$, and sends X to **MB**.
2. **MB** sends $\{(R_i, \sigma_i)\}_{i\in[n]}$ to **GW**.
3. Upon receiving $\{(R_i, \sigma_i)\}_{i\in[n]}$, **GW** does:
   (1) Check if σ_i is a valid signature on R_i using pk for $i \in [n]$; if not, halt and output $\perp$.
   (2) Choose a random $y \in \mathbb{Z}_p^*$ and compute $Y = g^y$. Compute $X_i = (R_i \cdot Y)^x = g^{xr_{w,i}+xk_i+xy}$ for $i \in [n]$, and return $\{X_i\}_{i\in[n]}$ to **MB**.
4. **MB** computes $K_i = X_i/(X)^{k_i} = g^{xr_{w,i}+xy}$ for $i \in [n]$ as the reusable randomized rule for rule r_i.
5. **GW** sets $I_0 = xy$, $I_1 = x$, $I_2 = g^w$ as the initialization parameters, and sends (I_0, I_1, I_2) to the clients within its domain.

Fig. 2. Preprocessing protocol

Input: The client (resp. the server) has input c. **MB** has input $\{K_i\}_{i\in[n]}$.

The protocol is run among a client, a server and **MB**:

1. The client computes $C = g^c$ and sends C to **MB** (through **GW**). Meanwhile, the server sets $C_s = c$ and sends C_s to **MB**.
2. **MB** checks whether C equals g^{C_s}. If yes, for $i \in [n]$, it calculates $S_i = (K_i \cdot C)^{C_s} = g^{c(xr_{w,i}+xy+c)}$ as the session detection rule for rule r_i.

Fig. 3. Session detection rule preparation protocol

encrypted TLS traffic as the client does. The server then checks whether the resulting encrypted tokens equal the encrypted tokens received from **MB**. If not, it indicates that the client is malicious. On the other hand, if it is the traffic sent from the server to the client, the client will do the same token encryption algorithm as the server does, and compares the resulting encrypted tokens with the received encrypted tokens from **MB** as well.

Rule Addition. In practice, new rules may be required to be added into the system. For a new rule $r'_i \in \mathcal{R}$ for $i \in [n']$, **RG** randomly chooses $k'_i \in \mathbb{Z}_p$, calculates $r'_{w,i} = \mathsf{PRF}_W(r'_i)$ and $R'_i = g^{r'_{w,i}+k'_i}$. It then signs the generated R'_i with sk to generate the signature σ'_i of R'_i. Finally, it sends the newly added attack rule tuples $\{R'_i, \sigma'_i, k'_i\}_{i\in[n']}$ to **MB**. For the newly added attack rule tuples, the rule addition protocol is described in Fig. 5, which is a simplified protocol of the preprocessing protocol.

Input: The client has inputs (I_0, I_1, I_2), a token t, the random keys k_s and c, the value R, a salt salt and a counter table T, where $I_0 = xy$, $I_1 = x$ and $I_2 = g^w$.

The algorithm is run by the client as follows:

1. Compute $I = I_0 + c = xy + c$.
2. For each token t:
   - If there exists no tuple corresponding to t in T: compute $t_w = \mathsf{PRF}_{I_2}(t)$, $T_t = g^{c(I_1 t_w + I)} = g^{c(xt_w+xy+c)}$, set $\mathsf{count}_t = 0$, compute the encryption of t as $E_t = \mathsf{Enc}_{T_t}(\mathsf{salt})$. Finally, insert tuple $(t, T_t, \mathsf{count}_t)$ into T.
   - If there exists a tuple $(t', T_{t'}, \mathsf{count}_{t'})$ in T where $t' = t$: update $\mathsf{count}_{t'} = \mathsf{count}_{t'} + 1$, and compute the encryption of t as $E_t = \mathsf{Enc}_{T_{t'}}(\mathsf{salt} + \mathsf{count}_{t'})$.
3. If it is the first session, compute $C_{ks} = g^{k_s}$, $C_w = (I_2)^{k_s} = g^{wk_s}$, $C_x = g^{I_1 k_s} = g^{xk_s}$ and $C_y = g^{I_0 k_s} = g^{xyk_s}$. The parameters $(\mathsf{salt}, C_{ks}, C_w, C_x, C_y)$ will be sent along with the encrypted token E_t for token t.

Fig. 4. Token encryption algorithm

Input: **MB** has newly added attack rule tuple set $\{(R'_i, \sigma'_i, k'_i)\}_{i\in[n']}$, where $R'_i = g^{r'_{w,i}+k'_i}$; **GW** has inputs Y, x.

The protocol is run between **GW** and **MB**:

1. **MB** sends $\{(R'_i, \sigma'_i)\}_{i\in[n']}$ to **GW**.
2. Upon receiving $\{(R'_i, \sigma'_i)\}_{i\in[n']}$, **GW** does: (1) Check if σ'_i is a valid signature on R'_i using pk for $i \in [n']$; if not, halt and output $\perp$. (2) Compute $X'_i = (R'_i \cdot Y)^x = g^{xr'_{w,i}+xk'_i+xy}$ for $i \in [n']$, and send $\{X'_i\}_{i\in[n']}$ to **MB**.
3. **MB** computes the reusable randomized rule $K'_i = X'_i/(X)^{k'_i}$ for $i \in [n']$.

Fig. 5. Rule addition protocol

5 Security

5.1 Middlebox Searchable Encryption

Definition. For a message space $\mathcal{M}$, a middlebox searchable encryption scheme consists of the following algorithms:

- $\mathsf{Setup}(\lambda)$: Takes a security parameter λ, outputs a key sk.
- $\mathsf{TokenEnc}(t_1, ..., t_n, \mathsf{sk})$: Takes a token set $\{t_i \in \mathcal{M}\}_{i\in[n]}$ and the key sk, outputs a set of ciphertexts $(\mathsf{c}_1, ..., \mathsf{c}_n)$ and a salt salt.
- $\mathsf{RuleEnc}(r, \mathsf{sk})$: Takes a rule $r \in \mathcal{M}$, the key sk, outputs an encrypted rule $\mathsf{e_r}$.
- $\mathsf{Match}(\mathsf{e_r}, (\mathsf{c}_1, ..., \mathsf{c}_n), \mathsf{salt})$: Takes an encrypted rule $\mathsf{e_r}$, ciphertexts $\{\mathsf{c}_i\}_{i\in[n]}$ and salt, outputs the set of indexes $\{\mathsf{ind}_i\}_{i\in[l]}$, where $\mathsf{ind}_i \in [n]$ for $i \in [l]$.

Correctness. We refer the reader to Appendix A for its definition.

Security. It is defined between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

- Setup. $\mathcal{C}$ runs $\mathsf{Setup}(\lambda)$ and obtains the key sk.
- $\mathsf{Challenge}$. $\mathcal{A}$ randomly chooses two sets of tokens $S_0 = \{t_{0,1}, ..., t_{0,n}\}$, $S_1 = \{t_{1,1}, ..., t_{1,n}\}$ from $\mathcal{M}$ and gives the two sets to $\mathcal{C}$. Upon receiving S_0 and S_1, $\mathcal{C}$ flips a random coin b, runs $\mathsf{TokenEnc}(t_{b,1}, ..., t_{b,n}, \mathsf{sk})$ to obtain a set of ciphertexts $(\mathsf{c}_1, ..., \mathsf{c}_n)$ and a salt salt. It then gives $(\mathsf{c}_1, ..., \mathsf{c}_n)$ and salt to $\mathcal{A}$.
- Query. $\mathcal{A}$ randomly chooses a set of rules $(r_1, ..., r_m)$ from $\mathcal{M}$ and gives the rules to $\mathcal{C}$. Upon receiving the set of rules, for $i \in [m]$, $\mathcal{C}$ runs $\mathsf{RuleEnc}(r_i, \mathsf{sk})$ to obtain encrypted rule e_{r_i}. $\mathcal{C}$ then gives the encrypted rules $\{\mathsf{e}_{r_i}\}_{i\in[m]}$ to $\mathcal{A}$.
- Guess. $\mathcal{A}$ outputs a guess b' of b.

Let $I_{0,i}$ be the index set that match r_i in S_0 and $I_{1,i}$ be the index set that match r_i in S_1. If $I_{0,i} = I_{1,i}$ and $b' = b$ for all i, we say that the adversary wins the above game. The advantage of the adversary in the game is defined as $\Pr[b' = b] - 1/2$.

Definition 2. *A middlebox searchable encryption scheme is secure if no PPT adversary has a non-negligible advantage in the game.*

Construction. The construction below captures the main structure from the security point of view.

- $\mathsf{Setup}(\lambda)$: Let PRF be a pseudorandom function. Generate $x, y, c, w \in \mathbb{Z}_p$, set (x, y, c, g^w) as the key.
- $\mathsf{TokenEnc}(t_1, ..., t_n, \mathsf{sk})$: Let salt be a random salt. For $i \in [n]$, do: (a) Let count be the number of times that token t_i repeats in the sequence t_1,...,t_{i-1}; (b) Calculate $t_{w,i} = \mathsf{PRF}_{g^w}(t_i)$, $T_{t_i} = g^{c(xt_{w,i}+xy+c)}$, $\mathsf{c}_i = H(T_{t_i}, \mathsf{salt} + \mathsf{count})$. Finally, the algorithm outputs $(\mathsf{c}_1, ..., \mathsf{c}_n)$ and salt.
- $\mathsf{RuleEnc}(r, \mathsf{sk})$: Compute $r_w = \mathsf{PRF}_{g^w}(r)$, $S = g^{c(xr_w+xy+c)}$, output $H(S)$.

Theorem 1. *Suppose H is a random oracle, the construction in Sect. 5.1 is a secure middlebox searchable encryption scheme.*

The proof of this theorem is provided in Appendix B.1.

5.2 Preprocessing Protocol

Definition. The preprocessing protocol is a two-party computation between **GW** and **MB**. Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be the process of the computation, where for every inputs (a, b), the outputs are $(f_1(a, b), f_2(a, b))$. In our protocol, the input of **GW** is x and the input of **MB** is a derivation of r, and only **MB** receives the output.

Security. The security requirements include: (a) **GW** should not learn the value of each rule; (b) **MB** cannot forge any new reusable randomized rule that is different from the reusable randomized rules obtained during the preprocessing protocol. Intuitively, the second requirement is satisfied if **MB** cannot obtain the value x. Since both of **GW** and **MB** are assumed to be semi-honest, we adopt the security definition with static semi-honest adversaries as in [6]. Let π be the two-party protocol for computing f, View_i^π be the ith party's view during the execution of π, and Output^π be the joint output of **GW** and **MB** from the execution of π. For our protocol, since f is a deterministic functionality, we adopt the security definition for deterministic functionality as shown below.

Definition 3. *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a deterministic functionality. We say that π securely computes f in the presence of static semi-honest adversaries if (a) Output^π equals $f(a, b)$; (b) there exist PPT algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ such that (1) $\{\mathcal{B}_1(a, f_1(a, b))\} \stackrel{c}{\equiv} \{\mathsf{View}_1^\pi(a, b)\}$, (2) $\{\mathcal{B}_2(b, f_2(a, b))\} \stackrel{c}{\equiv} \{\mathsf{View}_2^\pi(a, b)\}$, where $a, b \in \{0,1\}^*$ and $|a| = |b|$.*

Protocol. In Fig. 6, we provide a simplified protocol that outlines the main structure of the preprocessing protocol.

Lemma 1. *No computationally unbounded adversary can guess a rule r_i with probability greater than $1/|\mathcal{R}|$ with input R_i.*

The proof of this lemma is provided in Appendix B.2.

Theorem 2. *The preprocessing protocol securely computes f in the presence of static semi-honest adversaries assuming the DDH assumption holds.*

The proof of this theorem is provided in Appendix B.3.

Inputs: **GW** has inputs $x, y \in \mathbb{Z}_p$; **MB** has inputs $(\{R_i, k_i\}_{i\in[n]})$, where $R_i = g^{r_{w,i}+k_i}$.
The protocol is run between **GW** and **MB**:

1. **GW** computes $X = g^x$, and sends X to **MB**.
2. **MB** sends $\{R_i\}_{i\in[n]}$ to **GW**.
3. **GW** computes $X_i = (R_i \cdot g^y)^x$ for $i \in [n]$, and send $\{X_i\}_{i\in[n]}$ to **MB**.
4. **MB** computes $K_i = X_i/(X)^{k_i}$ as the reusable randomized rule for rule r_i.

Fig. 6. Simplified preprocessing protocol

5.3 Token Encryption

It captures the security requirement that **GW** cannot learn the underlying token when given an encrypted token.

Definition. For a message space $\mathcal{M}$, a token encryption scheme is as follows:

- $\mathsf{Setup}(\lambda)$: Takes as input a security parameter λ, outputs a secret key sk and the public parameters pk.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{sk}, t)$: Takes as input the public parameters pk, a secret key sk and a token $t \in \mathcal{M}$, outputs a ciphertext c.

Security. It is defined between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

- Setup: $\mathcal{C}$ runs $\mathsf{Setup}(\lambda)$ and sends the public parameters pk to $\mathcal{A}$.
- $\mathsf{Challenge}$: $\mathcal{A}$ randomly chooses two tokens t_0, t_1 from $\mathcal{M}$ and sends them to $\mathcal{C}$. $\mathcal{C}$ flips a random coin $b \in \{0, 1\}$, runs $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{sk}, t_b)$, and sends c to $\mathcal{A}$.
- Guess: $\mathcal{A}$ outputs a guess b' of b.

The advantage of an adversary is defined to be $\Pr[b' = b] - 1/2$.

Definition 4. *A token encryption scheme is secure if no PPT adversary has a non-negligible advantage in the security game.*

Construction. The construction presented below outlines the main structure from the security point of view.

- $\mathsf{Setup}(\lambda)$: Let PRF be a pseudorandom function. Choose random value $x, y, c, w \in \mathbb{Z}_p$, calculate $p_1 = g^c$, $p_2 = g^w$, $p_3 = x$ and $p_4 = y$. Finally, set c as sk and (p_1, p_2, p_3, p_4) as pk.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{sk}, t)$: Let salt be a random salt. Calculate $t_w = \mathsf{PRF}_{p_2}(t)$, $T_t = g^{c(xt_w+xy+c)}$, $\mathsf{c} = H(T_t, \mathsf{salt})$. Output c and salt.

Theorem 3. *Suppose H is a random oracle, the construction in Sect. 5.3 is a secure token encryption scheme.*

The proof of this theorem is provided in Appendix B.4.

5.4 Rule Hiding

It captures the security requirement that **MB** cannot learn the underlying rule when given an attack rule tuple (issued by **RG**).

Definition. For a message space $\mathcal{M}$, a rule hiding scheme is defined as follows:

- $\mathsf{Setup}(\lambda)$: Takes as input a security parameter λ, outputs a secret key sk and the public parameters pk.
- $\mathsf{RuleHide}(\mathsf{pk}, \mathsf{sk}, r)$: Takes as input the public parameters pk, a secret key sk and a rule $r \in \mathcal{M}$, outputs a hidden rule.

Security. The security definition for a rule hiding scheme is defined between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ as follows.

- Setup: $\mathcal{C}$ runs $\mathsf{Setup}(\lambda)$ and gives the public parameters to $\mathcal{A}$.
- $\mathsf{Challenge}$: $\mathcal{A}$ chooses two random rules r_0, r_1 from $\mathcal{M}$, and sends them to $\mathcal{C}$. Upon receiving r_0 and r_1, $\mathcal{C}$ flips a random coin b, runs $\mathsf{RuleHide}(\mathsf{pk}, \mathsf{sk}, r_b)$ and returns the resulting hidden rule to $\mathcal{A}$.
- Guess: $\mathcal{A}$ outputs a guess b' of b.

Construction.

- $\mathsf{Setup}(\lambda)$: Let PRF be a pseudorandom function. Choose random $k, w \in \mathbb{Z}_p$, set g^w as sk, k as pk.
- $\mathsf{RuleHide}(\mathsf{pk}, \mathsf{sk}, r)$: Calculate $r_w = \mathsf{PRF}_{\mathsf{sk}}(r)$, $R = g^{r_w + k}$, and output R.

Theorem 4. *Suppose* PRF *is a pseudorandom function, the construction in Sect. 5.4 is a secure rule hiding scheme.*

The proof of this theorem is provided in Appendix B.5.

6 Performance Evaluations

We investigate the performance of the network connection between a client and a server. Since PrivDPI perfoms better than BlindBox, we only present the comparison with PrivDPI. Let an *one-round connection* be a connection from the client to the server. The running time of a one-round connection reflects how fast a client can be connected to a server, and the communication cost captures the amount of overhead data need to be transferred for establishing this connection. Ideally, the running time for one-round connection should be as small as possible. The less running time it incurs, the faster a client can connect to a server. Similarly, it is desirable to minimize network communication overhead. We test the running time and the communication cost of one-round connection for our protocol and PrivDPI respectively. Our experiments are run on a Intel(R) Core i7-8700 CPU running at 3.20 Ghz with 8 GB RAM under 64bit Linux operating system. The CPU supports AES-NI instructions, where

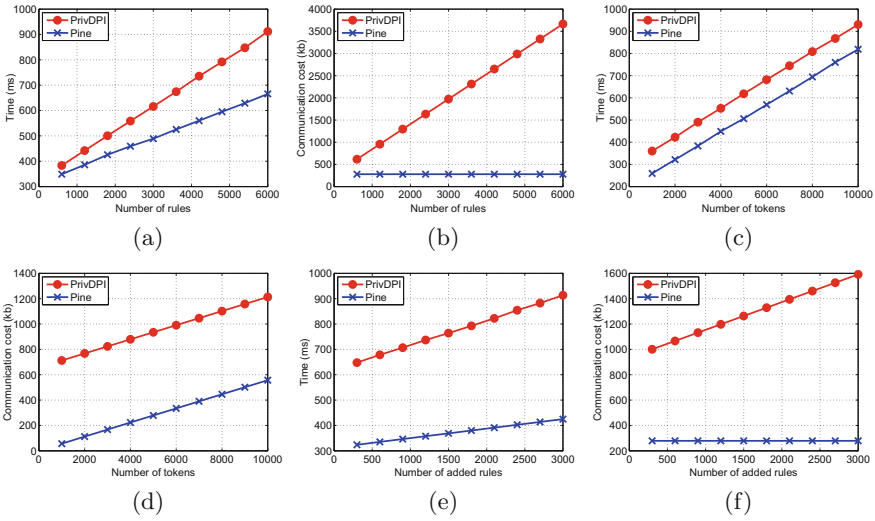

Fig. 7. Experimental performances

the encryption of token and the encryption of rule reflect this hardware support. The experiments are built on Charm-crypto [1], and is based on NIST Curve P-256. As stated in Sect. 3, both the rules and the tokens consist of 8 bytes. For simplicity, the payload that we test does not contain repeated tokens. We test each case for 20 times and takes the average.

How does the number of rules influence the one-round connection? Figure 7a illustrates the running time for one-round connection with 5,000 tokens when the number of rules range from 600 to 6,000. It is demonstrated that Pine takes less time than PrivDPI for each case, the more rules, the less time Pine takes compared to PrivDPI. This means that it takes less time for a client in Pine to connect to a server. In particular, for 5,000 tokens and 6,000 rules, it takes approximately 665 ms for Pine, while PrivDPI takes approximately 912 ms. That is, the delay for one-round connection of Pine is 27% less than PrivDPI; for 5,000 tokens and 3,000 rules, it takes approximately 488 ms for Pine, while PrivDPI takes approximately 616 ms. In other words, a client in Pine connects to a server with 20.7% faster speed than PrivDPI. Figure 7b shows the communication cost for one-round connection with 5,000 tokens when the number of rules range from 600 to 6,000. The communication cost of PrivDPI grows linearly with the number of rules, while for Pine it is constant. The more rules, the more communication cost PrivDPI incurs. This is because the client in PrivDPI needs to run the preprocessing protocol with **MB**, and the communication cost incurred by this preprocessing protocol is linear with the number of rules.

How does the number of tokens influence the one-round connection? We fix the number of rules to be 3,000, and test the running time and communication cost when the number of tokens range from 1,000 to 10,000.

Figure 7c shows that the running time of Pine is linear with the number of tokens in the payload, the same as PrivDPI. However, for each case, the time consumed of Pine is less than PrivDPI, this is due to the following two reasons. The first is that a client in Pine does not need to perform the preprocessing protocol for the 3,000 rules. The second is that, the encryption of a token in PrivDPI mainly takes one multiplication in G, one exponentiation in G, and one AES encryption. While in Pine, the encryption of a token mainly takes one hash operation, one exponentiation in G, and one AES encryption. That is, the token encryption of Pine is faster than that of PrivDPI. Figure 7d shows the communication cost of one-round connection with 3,000 rules when the number of tokens range from 1,000 to 10,000. Similar to the running time, the communication costs of Pine and PrivDPI are both linear with the number of tokens, but Pine incurs less communication than PrivDPI. This is due to the additional communication cost of the preprocessing protocol in PrivDPI for 3,000 rules.

How does the number of newly added rules influence the one-round connection? We test the running time and communication cost with 3,000 rules and 5,000 tokens when the number of newly added rules range from 300, to 3,000. Figure 7e shows that Pine takes less time than PrivDPI. For 3,000 newly added rules, Pine takes 424.96 ms, while PrivDPI takes 913.52 ms. That is, Pine is 53.48% faster than PrivDPI. Figure 7f shows that the communication cost of Pine is less than PrivDPI. In particular, the communication cost of Pine is independent of the number of newly added rules, while PrivDPI is linear with the number of newly added rules. This is because the client in Pine does not need to perform preprocessing protocol online.

7 Related Work

Our protocol is constructed based on BlindBox proposed by Sherry *et al.* [20] and PrivDPI proposed by Ning *et al.* [15], as was stated in the introduction. BlindBox introduces privacy-preserving deep packet inspection on encrypted traffic directly, while PrivDPI utilises an obfuscated rule generation mechanism with improved performance compared to BlindBox. Using the construction in BlindBox as the underlying component, Lan *et al.* [9] further proposed Embark that leverages on a trusted enterprise gateway to perform privacy-preserving detection in a cloud-based middlebox setting. In Embark, the enterprise gateway needs to be fully trusted and learns the content of the traffic and the detection rules, although in this case the client does not need to perform any operation as in our protocol. Our work focuses on the original setting of BlindBox and PrivDPI with further performance improvements, new properties and stronger privacy guarantee, while considering the practical enterprise gateway setting, in which the gateway needs not be fully trusted. Canard *et al.* [4] also proposed a protocol, BlindIDS, based on the concept of BlindBox, that has a better performance. The protocol consists of a token-matching mechanism that is based on pairing-based public key operation. Though practical, it is not compatible to TLS protocol.

Another related line of work focuses on accountability of the middlebox. This means the client and the server are aware of the middlebox that performs inspection on the encrypted traffic and are able to verify the authenticity of these middleboxes. Naylor *et al.* [14] first proposed such a scheme, termed mcTLS, where the existing TLS protocol is modified in order to achieve the accountability properties. However, Bhargavan *et al.* [3] showed that mcTLS can be tampered by an attacker to create confusion on the identity of the server that a middlebox is connected to, as well as the possibility for the attacker to inject its own data to the network. Due to this, a formal model on analyzing this type of protocols was proposed. Naylor *et al.* [13] further proposed a scheme, termed mbTLS, which does not modify the TLS protocol, thus allowing authentication of the middleboxes without needing to replace the existing TLS protocol. More recently, Lee *et al.* [10] proposed maTLS, a protocol that performs explicit authentication and verification of security parameters.

There are also proposals that analyse encrypted traffic without decrypting or inspecting the encrypted payloads. Machine learning models were utilised to detect anomalies based on the meta data of the encrypted traffic. Anderson *et al.* [2] proposed such techniques for malware detection on encrypted Traffic. Trusted hardware has also been deployed for privacy-preserving deep packet inspection. Most of the proposals utilize the secure enclave of Intel SGX. The main idea is to give the trusted hardware, resided in the middlebox, the session key. These include SGX-Box proposed by Han *et al.* [8], SafeBricks by Poddar *et al.* [19] and ShieldBox by Trach *et al.* [21] and LightBox by Duan *et al.* [5].

We note that our work can be combined with the accountability protocols, as well as the machine learning based works to provide comprehensive encrypted inspection that encompasses authentication and privacy.

8 Conclusion

In this paper, we proposed Pine, a protocol that allows inspection of encrypted traffic in a privacy-preserving manner. Pine builds upon the settings of BlindBox and techniques of PrivDPI in a practical setting, yet enables hiding of rule sets from the middleboxes with significantly improved performance compared to the two prior works. Furthermore, the protocol allows lightweight rules addition on the fly, which to the best of our knowledge has not been considered previously. Pine utilises the common practical enterprise setting where clients establish connections to Internet servers via an enterprise gateway, in such a way that the gateway assists in establishing the encrypted rule sets without learning the content of the client's traffic. At the same time, a middlebox inspects the encrypted traffic without learning both the underlying rules and content of the traffic. We demonstrated the improved performance of Pine over PrivDPI through extensive experiments. We believe Pine is a promising approach to detect malicious traffic amid growing privacy concerns for both corporate and individual users.

Acknowledgments. This work is supported in part by Singapore National Research Foundation (NRF2018NCR-NSOE004-0001) and AXA Research Fund, in part by

the National Natural Science Foundation of China (61822202, 61872089, 61902070, 61972094, 61825203, U1736203, 61732021), Guangdong Provincial Special Funds for Applied Technology Research and Development and Transformation of Key Scientific and Technological Achievements (2016B010124009), and Science and Technology Program of Guangzhou of China (201802010061), and in part by the National Research Foundation, Prime Ministers Office, Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd.

A Correctness of Middlebox Searchable Encryption

On one hand, for every token that matches a rule r, the match should be detected with probability 1; on the other hand, for a token that does not match r, the probability of the match should be negligibly small. For every sufficiently large security parameter λ and any polynomial $n(\cdot)$ such that $n = n(\lambda)$, for all $t_1, ..., t_n \in \mathcal{M}^n$, for each rule $r \in \mathcal{M}$, for each index ind_i satisfying $r = t_{\mathsf{ind}_i}$ and for each index ind_j satisfying $r \neq t_{\mathsf{ind}_j}$, let $\mathsf{Exp}_1(\lambda)$ and $\mathsf{Exp}_2(\lambda)$ be experiments defined as follows:

Experiment $\mathsf{Exp}_1(\lambda)$:

$$\mathsf{sk} \leftarrow \mathsf{Setup}(\lambda); (\mathsf{c}_1, ..., \mathsf{c}_n), \mathsf{salt} \leftarrow \mathsf{TokenEnc}(t_1, ..., t_n, \mathsf{sk}); \mathsf{e}_\mathsf{r} \leftarrow \mathsf{RuleEnc}(r, \mathsf{sk});$$
$$\{\mathsf{ind}_k\}_{k\in[l]} \leftarrow \mathsf{Match}(\mathsf{e}_\mathsf{r}, (\mathsf{c}_1, ..., \mathsf{c}_n), \mathsf{salt}) : \mathsf{ind}_i \in \{\mathsf{ind}_k\}_{k\in[l]}$$

Experiment $\mathsf{Exp}_2(\lambda)$:

$$\mathsf{sk} \leftarrow \mathsf{Setup}(\lambda); (\mathsf{c}_1, ..., \mathsf{c}_n), \mathsf{salt} \leftarrow \mathsf{TokenEnc}(t_1, ..., t_n, \mathsf{sk}); \mathsf{e}_\mathsf{r} \leftarrow \mathsf{RuleEnc}(r, \mathsf{sk});$$
$$\{\mathsf{ind}_k\}_{k\in[l]} \leftarrow \mathsf{Match}(\mathsf{e}_\mathsf{r}, (\mathsf{c}_1, ..., \mathsf{c}_n), \mathsf{salt}) : \mathsf{ind}_j \notin \{\mathsf{ind}_k\}_{k\in[l]}$$

We have $\Pr[\mathsf{Exp}_1(\lambda)] = 1$, $\Pr[\mathsf{Exp}_2(\lambda)] = \mathsf{negl}(\lambda)$.

B Proofs

B.1 Proof of Theorem 1

The security is proved via one hybrid, which replaces the random oracle with deterministic random values. In particular, the algorithm $\mathsf{TokenEnc}$ now is modified as follows: $\mathsf{Hybrid.TokenEnc}(t_1, ..., t_n, \mathsf{sk})$: Let salt be a random salt. For $i \in [n]$, sample a random value T_i in the ciphertext space and set $\mathsf{c_i} = \mathsf{T}_i$. Finally, output $(\mathsf{c}_1, ..., \mathsf{c}_n)$ and salt. The algorithm $\mathsf{RuleEnc}(r, \mathsf{sk})$ is defined to output a random value R from the ciphertext space with the restriction that: (1) if r equals t_i for some t_i, R is set to be T_i; (2) for any future r' such that r equals r', the output is set to be R. We have that the outputs of algorithm $\mathsf{TokenEnc}$ and algorithm $\mathsf{RuleEnc}$ are random, while the the pattern of matching between tokens and rules are preserved. Clearly, the distributions for $S_0 = \{t_{0,1}, ..., t_{0,n}\}$, $S_1 = \{t_{1,1}, ..., t_{1,n}\}$ are the same. Hence, any PPT adversary has a change of distinguishing the two sets of exactly half.

B.2 Proof of Lemma 1

Fix a random $R = g^r$, where $r \in \mathbb{Z}_p$. We have that the probability for $R = R_i$ is the probability for $k_i = r - r_{w,i}$. Hence, for $\forall R \in G$, $\Pr[R = R_i] = 1/p$.

B.3 Proof of Theorem 2

We construct a simulator for each of the parties, $\mathcal{B}_1$ for **GW** and $\mathcal{B}_2$ for **MB**. For the case when **GW** is corrupted, $\mathcal{B}_1$ needs to generate the view of the incoming messages for **GW**. The message that **GW** received is R_i for $i \in [n]$. To simulate R_i for rule r_i, $\mathcal{B}_1$ chooses a random $u_i \in \mathbb{Z}_p$, calculate $U_i = g^{u_i}$ and sets U_i as the incoming message for rule r_i which simulates the incoming message from **MB** to **GW**. Following Lemma 1, the distribution of the simulated incoming message for **GW** (i.e., U_i) is indistinguishable from a real execution of the protocol. We next consider the case when **MB** is corrupted. The first and the third messages are the incoming message that **MB** received. For the first message, $\mathcal{B}_2$ randomly chooses a value $v \in \mathbb{Z}_p$, computes $V = g^v$, and sets V as the first incoming message for **MB**. For the third message, $\mathcal{B}_2$ randomly chooses a value $v_i \in \mathbb{Z}_p$ for $i \in [n]$, computes $V_i = g^{v_i}$ and sets V_i as the incoming message during the third step of the protocol. The view of **MB** for a rule r_i in an real execution of the protocol is $(k_i, R_i; X, X_i)$. The distributions of the real view and the simulated view are $(k_i, g^{r_{w,i}+k_i}; g^x, g^{xr_{w,i}+xk_i+xy})$ and $(k_i, g^{r_{w,i}+k_i}; g^v, g^{v_i})$. Clearly, a PPT adversary cannot distinguish these two distributions if DDH assumption holds.

B.4 Proof of Theorem 3

We prove the security by one hybrid, where we replace the random oracle with random values. In particular, the modified algorithm Enc is described as follows: $\mathsf{Enc}(\mathsf{pk}, \mathsf{sk}, t)$: Let salt be a random salt. Sample a random value c^* from the ciphertext space, and output c^* and salt. Now we have that the output of algorithm Enc is random. The distributions for challenge tokens t_0 and t_1 are the same. Hence, there exists no PPT adversary that has a chance of distinguishing the two tokes of exactly half.

B.5 Proof of Theorem 4

We proof the security via one hybrid, which replaces the output of PRF with a random value. In particular, during the challenge phase, the challenger chooses a random value v, computes $R^* = g^{v+k}$ (where k is publicly known to the adversary), returns R^* to the adversary. Clearly, if the adversary wins the security game, one can build a simulator that utilizes the ability of the adversary to break the pseudorandom property of PRF.

References

1. Akinyele, J.A., et al.: Charm: a framework for rapidly prototyping cryptosystems. J. Cryptographic Eng. **3**(2), 111–128 (2013). https://doi.org/10.1007/s13389-013-0057-3
2. Anderson, B., Paul, S., McGrew, D.A.: Deciphering malware's use of TLS (without decryption). J. Comput. Virol. Hacking Tech. **14**(3), 195–211 (2018). https://doi.org/10.1007/s11416-017-0306-6
3. Bhargavan, K., Boureanu, I., Delignat-Lavaud, A., Fouque, P.A., Onete, C.: A formal treatment of accountable proxying over TLS. In: S&P 2018, pp. 339–356. IEEE Computer Society (2018)
4. Canard, S., Diop, A., Kheir, N., Paindavoine, M., Sabt, M.: BlindIDS: market-compliant and privacy-friendly intrusion detection system over encrypted traffic. In: AsiaCCS 2017, pp. 561–574. ACM (2017)
5. Duan, H., Wang, C., Yuan, X., Zhou, Y., Wang, Q., Ren, K.: Lightbox: full-stack protected stateful middlebox at lightning speed. In: CCS 2019, pp. 2351–2367 (2019)
6. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press (2009)
7. Google. HTTPS Encryption on the Web (2019). https://transparencyreport.google.com/https/overview?hl=en
8. Han, J., Kim, S., Ha, J., Han, D.: SGX-Box: enabling visibility on encrypted traffic using a secure middlebox module. In: APNet 2017, pp. 99–105. ACM (2017)
9. Lan, C., Sherry, J., Popa, R.A., Ratnasamy, S., Liu, Z.: Embark: securely outsourcing middleboxes to the cloud. In: NSDI 2016, pp. 255–273. USENIX Association (2016)
10. Lee, H., et al.: maTLS: how to make TLS middlebox-aware? In: NDSS 2019 (2019)
11. Meeker, M.: Internet trends (2019). https://www.bondcap.com/report/itr19/
12. National Security Agency. Managing Risk From Transport Layer Security Inspection (2019). https://www.us-cert.gov/ncas/current-activity/2019/11/19/nsa-releases-cyber-advisory-managing-risk-transport-layer-security
13. Naylor, D., Li, R., Gkantsidis, C., Karagiannis, T., Steenkiste, P.: And then there were more: secure communication for more than two parties. In: CoNEXT 2017, pp. 88–100. ACM (2017)
14. Naylor, D., et al.: Multi-context TLS (mcTLS): enabling secure in-network functionality in TLS. In: SIGCOMM 2015, pp. 199–212. ACM (2015)
15. Ning, J., Poh, G.S., Loh, J.C.N., Chia, J., Chang, E.C.: PrivDPI: privacy-preserving encrypted traffic inspection with reusable obfuscated rules. In: CCS 2019, pp. 1657–1670 (2019)
16. O'Neill, M., Ruoti, S., Seamons, K.E., Zappala, D.: TLS inspection: how often and who cares? IEEE Internet Comput. **21**(3), 22–29 (2017)
17. Paxson, V.: Bro: a system for detecting network intruders in real-time. Computer Netw. **31**(23–24), 2435–2463 (1999)
18. McAfee Network Security Platform (2019). http://www.mcafee.com/us/products/network-security-platform.aspx
19. Poddar, R., Lan, C., Popa, R.A., Ratnasamy, S.: SafeBricks: shielding network functions in the cloud. In: NSDI 2018, pp. 201–216. USENIX Association (2018)
20. Sherry, J., Lan, C., Popa, R.A., Ratnasamy, S.: BlindBox: deep packet inspection over encrypted traffic. In: SIGCOMM 2015, pp. 213–226 (2015)
21. Trach, B., Krohmer, A., Gregor, F., Arnautov, S., Bhatotia, P., Fetzer, C.: ShieldBox: secure middleboxes using shielded execution. In: SOSR 2018, pp. 2:1–2:14. ACM (2018)