PrivBox

Privacy-Preserving Deep Packet Inspection with Dual Double-masking Obfuscated Rule Generation

Wu, Pengfei; Ning, Jianting; Huang, Xinyi; Chen, Rongmao; Zhang, Kai; Liang, Kaitai

DOI
10.1109/TDSC.2025.3557423

Publication date
2025

Document Version
Final published version

Published in
IEEE Transactions on Dependable and Secure Computing

Citation (APA)
Wu, P., Ning, J., Huang, X., Chen, R., Zhang, K., & Liang, K. (2025). PrivBox: Privacy-Preserving Deep Packet Inspection with Dual Double-masking Obfuscated Rule Generation. *IEEE Transactions on Dependable and Secure Computing*, *22*(5), 4954-4970. https://doi.org/10.1109/TDSC.2025.3557423

PrivBox: Privacy-Preserving Deep Packet Inspection With Dual Double-Masking Obfuscated Rule Generation

Pengfei Wu, *Member, IEEE*, Jianting Ning, *Member, IEEE*, Xinyi Huang, Rongmao Chen, Kai Zhang, and Kaitai Liang, *Member, IEEE*

***Abstract*—Many network middleboxes have been deployed to perform *deep packet inspection* (DPI) over packet payloads. However, such middleboxes cannot accomplish their tasks when the traffic is encrypted. *BlindBox* (SIGCOMM 2015) provided the first solution for performing DPI over encrypted traffic. To improve its efficiency, a later proposal *PrivDPI* (CCS 2019) introduced a practical technique to generate encrypted rules. However, a recent proposal *P2DPI* (ASIACCS 2021) showed that the rule generator in PrivDPI can comprise the user's privacy. In this article, we present a new attack on P2DPI and show that the privacy of its endpoints can still be compromised by the rule generator. We comprehensively analyze the vulnerability of prior studies and present *PrivBox*, a new DPI system that achieves the same privacy guarantee as BlindBox while maintaining practical efficiency. This is based on a new technique called *dual double-masking obfuscated rule generation*. For a ruleset of 3,000, PrivBox achieves connection establishment time on the endpoint side comparable to PrivDPI and supports up to 4,672 token encryptions per second, which is sufficient for a number of real-world applications. Overall, our experiment demonstrates that PrivBox is practical and well-suited for short, frequently established sessions, especially when token repeating is common.**

Received 31 October 2023; revised 23 March 2025; accepted 28 March 2025. Date of publication 3 April 2025; date of current version 4 September 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62425205, Grant 62032005, Grant 62122092, and Grant 62372285, in part by Shanghai Rising-Star Program under Grant 22QA1403800, and in part by the EU Horizon Europe Research and Innovation Program under Grant 101073920 (TENSOR), Grant 101070052 (TANGO), and Grant 101070627 (REWIRE). *(Corresponding author: Jianting Ning.)*

Pengfei Wu is with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China, and also with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: pfwu@smu.edu.sg).

Jianting Ning is with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China, also with the Faculty of Data Science, City University of Macau, Macau 999078, China, and also with the College of Computer and Cyber Security, Fujian Normal University, Fuzhou 350007, China (e-mail: jtning88@gmail.com).

Xinyi Huang is with the College of Cyber Security, Jinan University, Guangzhou 510632, China (e-mail: xinyi@ust.hk).

Rongmao Chen is with the College of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: chromao@nudt.edu.cn).

Kai Zhang is with the College of Computer Science and Technology, Shanghai University of Electric Power, Shanghai 201399, China (e-mail: kzhang@shiep.edu.cn).

Kaitai Liang is with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 Delft, CD, The Netherlands (e-mail: Kaitai.Liang@tudelft.nl).

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2025.3557423, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2025.3557423

***Index Terms*—Traffic inspection, network privacy, privacy-preserving.**

I. Introduction

To PROTECT network operators and end users, network middleboxes have been deployed to perform deep packet inspection (DPI) to provide a wide range of functionalities, including intrusion detection, intrusion prevention, exfiltration prevention, etc. These middleboxes share a common feature that they inspect packet payloads directly. In recent years, however, the use of encryption protocols (such as HTTPS) has been a dramatic increase in traffic on the Internet. It is estimated that more than 80% of enterprises' web traffic is sent over HTTPS, and 94% of Google web traffic is encrypted [10]. Though these encryptions protect the confidentiality of network packets, they pose a long-standing challenge for DPI services. In particular, traditional network middleboxes can no longer perform DPI tasks since packet payloads are encrypted.

To inspect encrypted traffic, a well-known technique called "split-TLS" has been deployed in some middlebox systems. In these systems, a man-in-the-middle (MitM) attack on TLS is mounted and the traffic is decrypted (and inspected) at the middlebox. However, such MitM approach is not deemed to be secure: it violates the end-to-end security guarantees of TLS and hence various security issues have been caused (due to the weaknesses in its implementation) [11], [14], [20], [37]. The recent unfortunate set of issues prompted National Security Agency to outline the risks of Transport Layer Security Inspection [1] and the US-Cert to issue an alert (TA17-075 A) about problems on some HTTPS inspection products [36].

To enable DPI over encrypted traffic while preserving user privacy, new solutions have been proposed in recent years. These solutions need to deal with an unfortunate choice of only one of two desirable but seemingly conflicting properties: the protection of endpoint's privacy and the functionality of DPI. To address this problem, Sherry et al. [31] proposed *BlindBox*, demonstrating the possibility of building a system that satisfies both of these controversial properties. BlindBox performs DPI directly on the encrypted payload, keeping the rules hidden from the endpoints and payload hidden from the middlebox (except for the content that matches predefined rules). To that end, Sherry et al. introduced a technique named *obfuscated rule*

encryption that allows the middlebox to obtain encrypted rules generated from the private key of the endpoints and the rules from the middlebox in a privacy-preserving manner (i.e., without leaking the private key to the middlebox and the rules to the endpoints). The technique of obfuscated rule encryption is built on garbled circuit, which incurs significant computation and communication overheads (around 97 s for the setup as reported in [31]). To improve the overall efficiency, Ning et al. [27] introduced the technique of *reusable obfuscated rule generation* that significantly reduces the complexity. However, as pointed in [21], the threat model in PrivDPI does not include the case when the *rule generator* is compromised, and hence PrivDPI does not retain the same privacy as BlindBox. To address the scenario in which either the rule generator or the middlebox is coerced by the attacker (i.e., the same privacy guarantee as BlindBox), Kim et al. [21] recently proposed a DPI system called P2DPI. However, as we will demonstrate in a later section (see Section III), P2DPI still suffers an attack (that we propose in this paper) launched by the compromised rule generator, and hence cannot achieve the same privacy level as BlindBox (unlike what they claimed). In particular, the rule generator in P2DPI can generate more (new) obfuscated rules than allowed when the rule generator is compromised. We present an attack on P2DPI, demonstrating that it does not guarantee the claimed privacy. The reason why it suffers from our attack is that the generation of obfuscated rules in P2DPI is *malleable*. That is, given an obfuscated rule of a rule r, it is possible to generate a valid obfuscated rule of a rule r' for $r' \neq r$.

Keeping the efficiency limit on BlindBox as well as the privacy concern on PrivDPI and P2DPI in mind, we need to address the following two challenges:

Challenge 1: How does the middlebox interact with the rule generator securely and efficiently to prepare rule tuples? We investigate prior studies including PrivDPI and P2DPI and find that the main reason for the above vulnerability is that the rule generator has to initialize and know *all* secrets in the system setup. Upon obtaining an obfuscated rule from endpoints, it can easily derive a new one using these secrets. A potential solution is to split secrets into two parts and allow the rule generator and middlebox to initialize them separately, e.g., via secret sharing, thereby avoiding the rule generator dominating the generation of rule tuples. However, consider that the rule generator can generate illegal information that helps forge valid obfuscated rules in interacting with the middlebox, and a complicated two-party protocol would lead to heavy computation and communication burden. An issue would arise to ensure two entities securely and efficiently prepare rule tuples.

Challenge 2: How does the middlebox securely and efficiently interact with the endpoints while simultaneously supporting reusable obfuscated rules? This is a fundamental problem in designing a DPI scheme. That is, the rule provided by the middlebox and the private key provided by the endpoints are always kept secret during the interaction process. A straightforward solution falls in a two-party computation between two entities, for example, following the idea of BlindBox that leverages the garbled circuit to produce obfuscated rules. However, even though the secrecy of rules can be guaranteed, it loses the support for reusability. In each session, the middlebox and endpoints require a long delay for connection, not suitable for frequent session establishment. Therefore, taking rule tuples from the first challenge, a second challenge is to deal with the secure and efficient production of obfuscated rules and simultaneously allows for reusability across different sessions.

A. Our Contributions

In this paper, we propose *PrivBox*, a new protocol that enables privacy-preserving deep packet inspection over encrypted traffic directly. Our scheme offers the same privacy level as BlindBox while maintaining efficiency close to PrivDPI. The main contributions are summarized as follows.

- *Effective Attack on P2DPI:* We revisit P2DPI, and demonstrate that it is susceptible to an attack (that we propose in this paper) when the rule generator is coerced by an attacker. We say that a rule generator is coerced if it is not fully trusted and will try to generate valid obfuscated rules for any new rules on its preference by utilizing the messages it observes. These newly generated obfuscated rules give the rule generator more power than allowed to scan the encrypted traffic, enabling it to exhaustively search the contents of the encrypted traffic and hence compromise the endpoint's privacy. We present our analysis on the rule obfuscation scheme in P2DPI, and further show that it is possible to generate a valid obfuscated rule for a new rule r' by modifying a given obfuscated rule of rule r (where $r \neq r'$).
- *New solution for secure DPI over encrypted traffic:* We present PrivBox as a replacement for Blindbox and PrivDPI with which a compromised rule generator or middlebox alone cannot forge a valid obfuscated rule. To this end, we introduce a new technique called *dual double-masking obfuscated rule generation*, which is instantiated by the rule preparation protocol and preprocessing protocol. Besides providing the guarantee of *exact match privacy* (i.e., the middlebox can learn the position of an attack keyword occurs in a flow), PrivBox can also be extended to support *probable-cause privacy model* in the sense that the middlebox can reveal the contents of the encrypted traffic when the traffic is suspicious.
- *Novel structure supporting non-malleability and reusability*. The core of above dual double-masking obfuscated rule generation is the so-called *dual double-masking structure*. It means that the output of rule preparation protocol (i.e., rule tuple) and the output of preprocessing protocol (i.e., obfuscated rule) are in the form of $g^{aF_1(r_i)+bF_2(r_i)}$, where $F_1(r_i)$ and $F_2(r_i)$ are two functions on a given rule r_i and a, b are secrets. This structure supports two important properties: *non-malleability* and *reusability*. The former ensures that given the obfuscated rules of a rule set $\{r_i\}_{i\in[n]}$, anyone (including the rule generator or the middlebox alone) who does not know a and b simultaneously cannot forge a new valid obfuscated rule for any rule $r' \notin \{r_i\}_{i\in[n]}$. The latter enables the obfuscated rules generated in the previous session(s) to be reused across

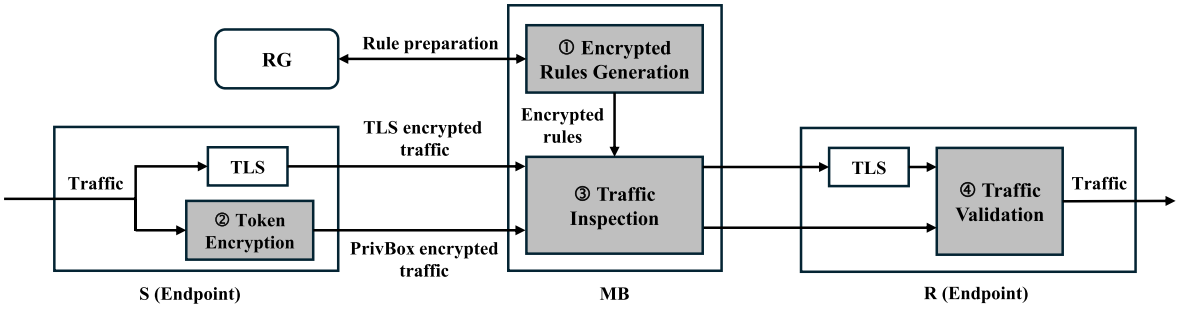


Fig. 1. System Architecture. **RG** first runs a rule preparation protocol with **MB**. **MB** then interacts with the endpoints to generate encrypted rules of the current session (①). For a payload sent through **MB**, in addition to the encryption of the traffic using regular TLS (i.e., *TLS encrypted traffic*), the sender encrypts the payload to generate the *PrivBox encrypted traffic* during the token encryption phase (②). **MB** performs inspection using the encrypted rules (generated from session rules) during the traffic inspection phase (③). Upon receiving the encrypted traffic, the receiver performs traffic validation in case the sender is malicious (④).

subsequent sessions, which speeds up the connection establishment. To further improve the performance of PrivBox, we introduce the property of reusability into the token encryption algorithm such that the intermediate values generated in each session can be reused within the same session or across different session(s) for repeated tokens. On top of this structure, our rule preparation protocol and preprocessing protocol are quite efficient, which only involve some lightweight group operations and verifications, so we believe this structure may have independent interests.

- *Extensive experiments for evaluating performance:* We conduct extensive experiments to demonstrate the performance of PrivBox through comparison with PrivDPI (and other related systems). The connection establishment time (i.e., preprocessing) in PrivBox on the endpoint side is close to PrivDPI. For 10 consecutive sessions, the connection establishment time in PrivBox is only 45.605 ms.[1] In our implementation, the token encryption rate of PrivBox is around 4,672 per second, which is sufficient for a number of real-world applications. By reusing the intermediate values generated in each session for repeated tokens, the token encryption delay of PrivBox can be further reduced. In particular, the performance of token encryption in PrivBox outperforms PrivDPI when the percentage of a token repeating exceeds around 93% for a set of 800 tokens. For a round trip on a ruleset of 3,000 in the first session, PrivBox is approximately 79.5% slower than PrivDPI for a set of 10,000 tokens. For a subsequent session under the same setting, PrivBox is approximately 65.6% slower than PrivDPI. This is because PrivBox employs dual double-masking obfuscated rule generation, which retains the same privacy guarantee as BlindBox (while PrivDPI cannot) but involves more computation on the group. For a set of 8,000 tokens with more than 92% already appeared in previous session(s), PrivBox runs faster than PrivDPI. To sum up, PrivBox retains efficiency close to PrivDPI but achieves the same privacy guarantee as BlindBox. Generally, PrivBox is more suitable for short, frequently established sessions. For those scenarios when token repeating frequently happens (e.g., more than 92% repeated tokens exist for a set of 8,000 tokens), PrivBox is also more practical than PrivDPI.

[1]The endpoint side is more sensitive to such delay since it is usually deployed on personal computers or mobile phones while the middlebox is generally deployed in the cloud with more powerful computing resources.

II. Overview

A. System Architecture

The system architecture of PrivBox is shown in Fig. 1, which follows BlindBox and PrivDPI. Specifically, there are four parties in PrivBox: sender (**S**), receiver (**R**), rule generator (**RG**) and middlebox (**MB**), which are described as follows.

- *S:* The endpoint that sends network traffic to **R**.
- *R:* The endpoint that receives network traffic from **S**.
- *RG:* It generates the rule tuples with **MB** that contain attack rules. Each rule aims to describe an attack and contains one or more keywords. These rule tuples will be utilized by **MB** to generate encrypted rules for inspection. The role of **RG** is usually performed by organizations that provide network security services, such as McAfee [28], Emerging Threats [33].
- *MB:* It is a network appliance that inspects the encrypted traffic sent between **S** and **R**. In particular, it tries to discover malicious payload in the traffic.

B. Usage Scenarios

Before presenting the system requirements and threat model of PrivBox, we describe the usage scenarios with the following examples. For each party in the examples, we will indicate the entity (i.e., **S**, **R**, **RG** and **MB**) in our model that corresponds to.

- *Cybercrime Examination:* Considering the increasing prevalence of financial fraud and money laundering, a multinational corporation is asked to register for data traffic detection with the internet service provider (**MB**) so that all its sent and received information can be monitored. It trusts the international criminal police organization (**RG**) which generates rulesets for monitoring. Each employee (**S** or **R**) of the corporation installs PrivBox HTTPS configuration, enabling her/his traffic to be scanned with the rules generated by **RG**. In this example, the corporation wants

MB to perform traffic detection, but **MB** is not allowed to obtain any information about the content of the traffic (as it is highly related to business confidentiality).

- *Phishing Detection:* Tom (**S** or **R**) is a student at a college. It is required by the college policy that all students' emails should be scanned and monitored. To get rid of phishing emails, Tom desires such a policy applied to his email. A security organization such as McAfee (**RG**) provides such a service that issues rules to the email server deployed by the college (**MB**). Tom allows **MB** to examine his emails but does not want it (or who could access **MB**) to read his email contents. To this end, Tom installs PrivBox HTTPS configuration such that **MB** can scan his traffic without revealing the contents.
- *Copyright Protection:* Suppose Carl (**R**) is a subscriber of a website (**S**) and all videos and pictures downloaded from the website are paid. In order to thwart some users from using bugs on website service to obtain videos and pictures without paying them, the official copyright administrator (**RG**) registers owned multimedia data with the internet service provider (**MB**), who is responsible for monitoring network traffic to filter out those flow downloading pirates. Therefore, Carl installs PrivBox HTTPS configuration as a plugin for downloading software such that **MB** can scan his traffic without knowing the downloaded contents.

C. System Requirements

The main goal of PrivBox is to perform inspection directly over encrypted traffic (i.e., without decrypting the encrypted traffic at **MB**) communicated between **S** and **R**. Specifically, there are five goals: (1) **MB** is able to check whether the encrypted traffic communicated between **S** and **R** is malicious; (2) The endpoints cannot learn what **MB** is detecting, i.e., cannot learn the rules; (3) If the encrypted traffic between the endpoints is not suspicious, then **MB** cannot read the contents; (4) Retain the same privacy level of BlindBox; (5) Preserve efficiency close to PrivDPI.

D. Threat Model

We consider three types of attackers (that reflect the same privacy guarantee as BlindBox) described as follows.

- *Type-I*: This type of attacker models as those in traditional intrusion detection system (IDS). Such attackers aim to escape the detection of malicious activities. The goal of PrivBox is to detect them over encrypted traffic. Similar to standard IDS, either **S** or **R** can be malicious, however, at least one of the endpoints should be honest[2]. This assumption is a standard setting in reality such as exfiltration detection and parental filtering [31].
- *Type-II*: This models the attacker at **MB**. Such an attacker attempts to subvert the system via trying to extract sensitive information from the encrypted traffic sent between the endpoints. **MB** is assumed to be semi-honest that it follows the system honestly but tries to learn the plaintext of the encrypted traffic and thus violates the privacy of **S** or **R** [9], [39], [40]. PrivBox aims to protect the privacy of the endpoints from **MB**, while enabling **MB** to perform DPI. Sharing the same philosophy with BlindBox and PrivDPI, we do not hide the rules from **MB**.
- *Type-III*: This models the attacker at **RG** for the case where **RG** is compromised. Such an attacker tries to extract private information from the messages it collects. In particular, **RG** is assumed to be malicious which may deviate from the protocol interacting with **MB** to learn its secrets. Besides, we allow **RG** to eavesdrop (but not manipulate) the data traffic over the network (between endpoints and between endpoints and **MB**) and attempt to forge an obfuscated rule that is illegal for the system. Such rules will further be used to inspect encrypted traffic to violate the privacy of the endpoints. Here, allowing a third party to access transmitted data is possible because the encrypted data can be observed by anyone including **RG**, which has also been considered in many secure data transmission solutions, e.g., [12], [21], [38]. Similar to BlindBox, we assume that at most one of **RG** and **MB** can be compromised. This is well understood as if both of them are controlled by an adversary, **RG** can generate any rules as it wishes and **MB** can use these rules to scan the traffic and further read all contents between endpoints.

E. System Flow

PrivBox consists of the following phases:

- *Setup:* **RG** runs a rule preparation protocol with **MB**. The endpoints install PrivBox HTTPS configuration, and derive keys from the session key established during the regular TLS handshake protocol. This phase is executed before any session starts.
- *Preprocessing:* **MB** runs a preprocessing protocol with the endpoints to generate the obfuscated rules. This preprocessing protocol is executed in the first session right after the completion of the regular TLS handshake protocol.
- $Session\ Rule\ Preparation:$ The obfuscated rules are used to generate the session rules for current session. Such session rules will be used to generate encrypted rules for traffic inspection in the subsequent inspection phases.
- $Token\ encryption:$ **S** generates a ciphertext from a payload[3] and sends it to **MB** for inspection before it is transmitted to **R**. Throughout this paper, we use the term encrypted traffic (or *PrivBox encrypted traffic*) to refer to the ciphertext mentioned above unless otherwise stated.
- $Traffic\ Inspection:$ **MB** performs packet inspection over the encrypted traffic it receives. If the traffic is malicious, **MB** takes the corresponding actions; otherwise, the non-matching traffic remains unreadable to **MB**.
- $Traffic\ Validation:$ This phase is executed on **R** to prevent **S** from sending not valid traffic. In particular, it

[2]Note that this is the fundamental requirement of IDS. This is because: if both **S** and **R** are malicious, they may agree on a secret key and encrypt the traffic sent between them with this key. If this happens, the encrypted traffic cannot be detected anymore.

[3]This payload is the same as the one encrypted in the standard TLS protocol.

first recovers a payload from the TLS encrypted traffic, and then runs the same token encryption algorithm as used by **S** to generate a new encrypted traffic. Recall that, in our threat model, either **S** or **R** may be malicious – this endpoint could try to cheat by not encrypting the tokens correctly or by encrypting only a subset of the tokens to eschew detection at **MB**. Since at least one endpoint is honest, such verification will prevent this attack. If the newly generated encrypted traffic does not equal to the traffic it receives, it aborts the session.

III. Attack on P2DPI

In this section, we analyze the security of P2DPI via presenting a new attack that enables a *compromised* **RG** to generate more (new) valid obfuscated rule(s) than allowed (on its own). As described in Section II-D, we say that **RG** (resp. **MB**) is compromised if **RG** (resp. **MB**) is not fully trusted but is curious about the contents of the encrypted traffic by collecting all message it observes. As stated in [21], [31], PrivBox cannot work in the setting where both **MB** and **RG** are compromised, but works for the case where either **RG** or **MB** is compromised.

A. Security Analysis on P2DPI

We present our attack on the rule setup stage of P2DPI when **RG** is compromised. We first revisit its rule setup stage:

1. **RG** generates a randomization key $k_{MB} \in \mathbb{Z}_p$, which is securely shared with **MB**, and obfuscates r_i to R_i with k_{MB} as follows:
   For $i \in [N_r]$, compute $R_i = (g^{H_1(r_i)})^{k_{MB}}$, where N_r is the total number of detection rules, g is the generator of a group, and $H_1(\cdot)$ is a hash function. **RG** then signs R_i to $\mathsf{sig}(R_i)$ using its private key and sends a set of detection rules $\{R_i, \mathsf{sig}(R_i) | i \in [N_r]\}$ to **MB**.
2. **S** and **R** send a connection request to **MB**.
3. **MB** sends $\{R_i, \mathsf{sig}(R_i) | i \in [N_r]\}$ to **S** and **R**.
4. Both **S** and **R** verify $\mathsf{sig}(R_i)$ using the **RG**'s public key. If they are valid, either **S** or **R** generates k_{SR} and shares it with the other using key sharing protocol of SSL/TLS handshake. Otherwise, they output $\perp$.
5. Both **S** and **R** compute a set of intermediate obfuscated rules $\{I_i = R_i^{k_{SR}} = (g^{H_1(r_i)})^{k_{MB} \cdot k_{SR}} | i \in [N_r]\}$. **S** and **R** return $\{I_i | i \in [N_r]\}$ to **MB**.
6. **MB** checks whether $\{I_i | i \in [N_r]\}$ sent from **S** and **R** are identical. If not, it disconnects the connection between **S** and **R**. If they are identical, **MB** computes a set of session rules $\{S_i = (I_i)^{1/k_{MB}} = g^{H_1(r_i) \cdot k_{SR}} | i \in [N_r]\}$.

As shown in Step 1, **RG** knows k_{MB} and r_i. **RG** can further obtains $I_i = (g^{H_1(r_i)})^{\cdot k_{MB} \cdot k_{SR}}$ in Step 5. For any new rule r^*, **RG** can forge the session rule of r^* by computing $S^* = (I_i)^{H_1(r^*)/(H_1(r_i) \cdot k_{MB})} = g^{H_1(r^*) \cdot k_{SR}}$. This shows that P2DPI is insecure against the Type-III attacker. **RG** can further exhaustively search the contents of the encrypted traffic that match the rule(s) by forging arbitrary session rule, thereby breaking the privacy of the encrypted traffic.

B. Discussion

We here give the reason why P2DPI cannot be secure against the Type-III attacker. We found that $I_i = (g^{H_1(r_i)})^{k_{MB} \cdot k_{SR}}$ for rule r_i (in Step 5 of the rule setup stage) in P2DPI is *malleable*. That is, given I_i, it is possible for **RG** to generate a modified I' that is *well-formed* for a new rule r'. We say that I' is well-formed if I' is in the form of $(g^{H_1(r')})^{k_{MB} \cdot k_{SR}}$.

To resist the above attack, we need to introduce an important property called *non-malleability* into the rule obfuscation scheme. Loosely speaking, a rule obfuscation scheme is *non-malleable* if the adversary can only output an invalid obfuscated rule for rule r' when given a valid obfuscated rule for rule r where $r \neq r'$.

IV. PrivBox

In this section, we present PrivBox, which is described right after the preliminaries. The rationale behind the rule preparation protocol in the setup phase and the preprocessing protocol in the preprocessing phase is a new technique called *dual double-masking obfuscated rule generation*. In particular, the obfuscated rule generated by these protocols has two important properties, namely *non-malleability* and *reusability*. The non-malleability property is served for security in the sense that anyone (including **MB** or **RG** alone) cannot generate a new valid obfuscated rule for $r^* \notin \{r_i\}_{i \in [n]}$ when given the obfuscated rules of $\{r_i\}_{i \in [n]}$. The property of reusability is used for efficiency in the sense that the obfuscated rule generated in previous session(s) can be reused across subsequent sessions and hence we can reduce the connection establishment delay.

A. Preliminaries

Let λ be the security parameter, PPT be probabilistic polynomial-time, and $\mathsf{R} = \{r_i\}_{i \in [n]}$ be the rule domain, where $[n]$ denotes the set $\{1, 2, \ldots, n\}$ and n denotes the number of rules in R. Similar to the setting in BlindBox and PrivDPI, both $\mathbf{RG}$ and $\mathbf{MB}$ know R. Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a data encapsulation mechanism with keyspace $\mathcal{K}$. We say Π is CPA-secure if it has indistinguishable encryptions under a chosen-plaintext attack. The definitions of the data encapsulation mechanism and the discrete logarithm assumption are given in Appendix A, available online. Let G be a group of prime order p and g be a generator of G. Define three hash functions H_1, H_2, H_3 as follows: $H_1 : G \to \mathcal{K}$, $H_2 : \mathsf{R} \to \mathbb{Z}_p$, $H_3 : G \to \mathbb{Z}_p$. We further define a "random function" H_4 that takes a random salt and a value in $\mathbb{Z}_p$, and outputs a random value. H_4 is pseudorandom and not invertible, which is implemented with AES algorithm. We additionally define a value RS, which is a parameter to reduce the ciphertext size such that the bandwidth overhead can be reduced. The notations used in the system are listed as follows:

- $(\widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), Sig_{MB}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i), Sig_{MB}(\widehat{R}_i))$: rule tuple of rule r_i, used to generate the obfuscated rule of r_i.
- K_i: obfuscated rule of r_i, used to generate the session rule of r_i.
- I_i: session rule of r_i, used to generate the encrypted rule.

Input: **RG** has inputs $(\{r_i\}_{i\in[n]}, a, r, H_1, H_2)$ and the public/private key pair $(pk_{Sig_{RG}}, sk_{Sig_{RG}})$ of Sig_{RG}. **MB** has inputs $(\{r_i\}_{i\in[n]}, b, s, y, \widetilde{y})$ and the public/private key pair $(pk_{Sig_{MB}}, sk_{Sig_{MB}})$ of Sig_{MB}.

Protocol:

1. **RG** calculates $S_A = g^a$, $L = g^r$, and sends (S_A, L) to **MB**.
2. **MB** calculates $S_B = g^b$, $S = g^s$, and sends (S_B, S) to **RG**.
3. **RG** calculates $\widetilde{S}_A = (S_B)^a = g^{ab}$, $V_i = (\widetilde{S}_A)^{H_2(r_i)} = g^{abH_2(r_i)}$ for $i \in [n]$, and sends $(\{V_i\}_{i\in[n]})$ to **MB**.
4. **MB** calculates $\widetilde{S}_B = (S_A)^b$ and checks whether the equation $V_i \stackrel{?}{=} (\widetilde{S}_B)^{H_2(r_i)}$ holds for $i \in [n]$. If not, it halts and aborts. Otherwise, it calculates $Y = \mathsf{Enc}_{H_1(\widetilde{S}_B)}(y||\widetilde{y})$, $\widetilde{R} = (\widetilde{S}_B)^s$, $S_i = V_i^s$ for $i \in [n]$, and sends them to **RG**.
5. **RG** runs $\mathsf{Dec}_{H_1(\widetilde{S}_A)}(Y)$ to obtain $(y, \widetilde{y})$, computes $R = (\widetilde{R})^r$, $\widehat{R} = R \cdot \widetilde{S}_A$, $R_i = (S_i)^r$, $\widetilde{R}_i = (R_i)^y \cdot g^{y\widetilde{y}\cdot H_3(R_i)}$, $\widehat{R}_i = g^{y\cdot H_3(R_i)}$ for $i \in [n]$, signs R, $\widetilde{R}_i$, $\widehat{R}_i$ with $sk_{Sig_{RG}}$ to obtain the signatures $Sig_{RG}(R)$, $\{Sig_{RG}(\widetilde{R}_i), Sig_{RG}(\widehat{R}_i)\}_{i\in[n]}$. It then sends $(\widehat{R}, Sig_{RG}(R), \{R_i, \widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i)\}_{i\in[n]})$ to **MB**, where $R = g^{absr}$, $R_i = g^{absrH_2(r_i)}$, $\widetilde{R}_i = g^{yabsrH_2(r_i)+y\widetilde{y}H_3(g^{absrH_2(r_i)})}$, $\widehat{R}_i = g^{yH_3(g^{absrH_2(r_i)})}$.
6. For $i \in [n]$, **MB** computes $R = \widehat{R}/\widetilde{S}_B$ and checks whether
   1) $Sig_{RG}(R)$ is a valid signature of R, $Sig_{RG}(\widetilde{R}_i)$ is a valid signature of $\widetilde{R}_i$ and $Sig_{RG}(\widehat{R}_i)$ is a valid signature of $\widehat{R}_i$ for $i \in [n]$;
   2) $e(R, g) \stackrel{?}{=} e(\widetilde{R}, L)$ holds;
   3) $e(L, S_i) \stackrel{?}{=} e(g, R_i)$, $\widetilde{R}_i \stackrel{?}{=} (R_i)^y \cdot g^{y\widetilde{y}H_3(R_i)}$, $\widehat{R}_i \stackrel{?}{=} R_i \cdot g^{yH_3(R_i)}$ hold for $i \in [n]$.

   If not, it halts and aborts. Otherwise, it signs R, $\{\widetilde{R}_i, \widehat{R}_i\}_{i\in[n]}$ with $sk_{Sig_{MB}}$ to obtain the signatures $Sig_{MB}(R)$, $\{Sig_{MB}(\widetilde{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]}$, and finally stores $(y, \widetilde{y}, R, Sig_{RG}(R), Sig_{MB}(R), \{\widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), Sig_{MB}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]})$ for future use.

Fig. 2. Rule preparation protocol.

- E_{r_i}: encrypted rule of r_i, used for traffic inspection.
- T_i: session token of token t_i, used to generate encrypted token.
- D_{t_i}: encrypted token of token t_i, used for traffic inspection.

B. Setup

RG first chooses a signature scheme Sig_{RG} with a public/private key pair $(pk_{Sig_{RG}}, sk_{Sig_{RG}})$, and chooses uniform $a, r \in \mathbb{Z}_p$ as its secret. Similarly, **MB** chooses a signature scheme Sig_{MB} with a public/private key pair $(pk_{Sig_{MB}}, sk_{Sig_{MB}})$, and chooses uniform $b, s, y, \widetilde{y} \in \mathbb{Z}_p$ as its secret. **RG** then runs the rule preparation protocol with **MB**, which is described in Fig. 2. This protocol is run before (and independent of) the connection between the endpoints. The aim of this protocol is to generate the *rule tuple* for each rule $r_i \in \mathsf{R}$, where $i \in [n]$. These rule tuples will be used to generate the obfuscated rules during the subsequent phase.

The rationale behind this protocol is as follows. To "securely generate" the rule tuple of each rule for **MB**, **RG** and **MB** engage in this rule preparation protocol. Here, "securely generate" means that anyone (including **MB** or **RG** alone) cannot forge the valid rule tuple for any rule $r^* \notin \{r_i\}_{i\in[n]}$ that will be accepted by the endpoints during the subsequent phase. Specifically, **RG** computes $S_A = g^a$ and $L = g^r$ using its secret (a, r), and sends (S_A, L) to **MB**. Here, L is served as a commitment of **RG**'s secret r. **MB** computes $S_B = g^b$, $S = g^s$, and sends (S_B, S) to **RG**. Similarly, S is served as a commitment of **MB**'s secret s. In the above two steps, S_A, S_B can be treated as the public keys of **MB** and **RG**, respectively. With **RG**'s public key S_A, **MB** can derive the shared secret key g^{ab} using its randomness b. Similarly, with **MB**'s public key S_B, **RG** can derive the shared secret key g^{ab} using its randomness a. Upon receiving (S_B, S) from **MB**, **RG** computes $\widetilde{S}_A = (S_B)^a$ (i.e., the shared secret key g^{ab}) using its randomness a, $V_i = (\widetilde{S}_A)^{H_2(r_i)}$ for rule r_i, and returns $\{V_i\}_{i\in[n]}$ to **MB**. Here, the shared secret key g^{ab} is served as a mask for $H_2(r_i)$ such that one cannot infer r_i once it sees V_i.

Upon receiving $\{V_i\}_{i\in[n]}$, **MB** computes $\widetilde{S}_B = (S_A)^b$ (i.e., the shared secret key g^{ab}) using its randomness b, and checks if the equation $V_i = (\widetilde{S}_B)^{H_2(r_i)}$ holds for $i \in [n]$ to ensure that V_i is honestly generated by **RG** using g^{ab} and $H_2(r_i)$. If $\{V_i\}_{i\in[n]}$ are correctly generated, **MB** attaches its secret s to $\widetilde{S}_B$ and V_i for $i \in [n]$ respectively to generate $\widetilde{R}$ and S_i for $i \in [n]$. Additionally, **MB** utilizes a data encapsulation mechanism to "securely transfer" $(y, \widetilde{y})$ to **RG**. Once receiving $(Y, \widetilde{R}, \{S_i\}_{i\in[n]})$ from **MB**, **RG** decrypts Y to obtain $(y, \widetilde{y})$, computes $R = (\widetilde{R})^r$, $\widehat{R} = R \cdot \widetilde{S}_A$, $R_i = (S_i)^r$, $\widetilde{R}_i = (R_i)^y \cdot g^{y\widetilde{y}\cdot H_3(R_i)}$, $\widehat{R}_i = g^{y\cdot H_3(R_i)}$ for $i \in [n]$, signs R, $\widetilde{R}_i$, $\widehat{R}_i$ with $sk_{Sig_{RG}}$ to obtain the corresponding signatures, and finally sends $(\widehat{R}, \{R_i, \widetilde{R}_i, \widehat{R}_i\}_{i\in[n]})$ and the corresponding signatures to **MB**. **MB** then computes $R = \widehat{R}/\widetilde{S}_B$ and checks whether $e(R, g) \stackrel{?}{=} e(\widetilde{R}, L)$ (resp. $e(L, S_i) \stackrel{?}{=} e(g, R_i)$) holds to ensure R (resp. R_i) is correctly generated from $\widetilde{R}$ (resp. S_i) and **RG**'s secret r. In addition, **MB** checks whether $\widetilde{R}_i$ and $\widehat{R}_i$ are correctly generated from R_i and $(y, \widetilde{y})$ by checking whether $\widetilde{R}_i \stackrel{?}{=} (R_i)^y \cdot g^{y\widetilde{y}H_3(R_i)}$ and $\widehat{R}_i \stackrel{?}{=} R_i \cdot g^{yH_3(R_i)}$ hold. If yes, it finally signs R, $\{\widetilde{R}_i, \widehat{R}_i\}_{i\in[n]}$ with $sk_{Sig_{MB}}$ to obtain the signatures $Sig_{MB}(R)$, $\{Sig_{MB}(\widetilde{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]}$.

Input: The endpoints have common inputs $pk_{Sig_{RG}}$ and $pk_{Sig_{MB}}$, k_{s_1}, k_{s_2}, K_{s_1}, where $K_{s_1} = g^{k_{s_1}}$; **MB** has inputs $(y, \tilde{y}, R, Sig_{RG}(R), Sig_{MB}(R), \{R_i, \widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), Sig_{MB}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]}, H_3)$, where $R = g^{absr}$, $R_i = g^{absrH_2(r_i)}$, $\widetilde{R}_i = g^{yabsrH_2(r_i)+y\tilde{y}H_3(g^{absrH_2(r_i)})}$, $\widehat{R}_i = g^{yH_3(g^{absrH_2(r_i)})}$.

Protocol:

1. Each endpoint sends its K_{s_1} to **MB**.
2. **MB** checks if the received K_{s_1} from the endpoints are the same. If not, it halts and aborts. Otherwise, it sends $(R, Sig_{RG}(R), Sig_{MB}(R), \{\widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), Sig_{MB}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]})$ to the endpoints.
3. Each endpoint works as follows:
   (1) For $i \in [n]$, check if $Sig_{RG}(R), Sig_{MB}(R), \{Sig_{RG}(\widetilde{R}_i), Sig_{MB}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]}$ are valid using $pk_{Sig_{RG}}$ and $pk_{Sig_{MB}}$. If not, halt and abort.
   (2) Calculate $K_i = (\widetilde{R}_i)^{k_{s_1}} \cdot (\widehat{R}_i)^{k_{s_2}}$ for $i \in [n]$, return $\{\widetilde{K}_i\}_{i\in[n]}$ to **MB**, and store R for future use.
4. **MB** checks whether the received $(\{\widetilde{K}_i\}_{i\in[n]})$ from the endpoints are the same. If not, it halts and aborts. Otherwise, it calculates $K_i = (\widetilde{K}_i/(K_{s_1})^{y\tilde{y}H_3(R_i)})^{1/y} = g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})}$ for $i \in [n]$. Finally, **MB** stores $(\{K_i\}_{i\in[n]})$ for future use.

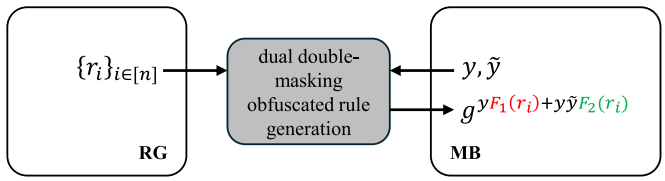
Fig. 3. Preprocessing protocol.

In this phase, the endpoints independently install a PrivBox HTTPS configuration which includes $H_1, H_2, H_3, H_4, pk_{Sig_{RG}}$ and $pk_{Sig_{MB}}$. They then run the regular TLS handshake protocol to agree on a session key k and derive three types of secret keys from k, described as follows: (1) k_{SSL}, the regular SSL key, used for encrypting the traffic; (2) $k_{s_1}, k_{s_2} \in \mathbb{Z}_p$, used to generate obfuscated rules during the preprocessing phase and the encrypted token during the token encryption phase; (3) k_r, a seed that will be used to generate randomness and the initial salt S_{salt}. The endpoints calculate $K_{s_1} = g^{k_{s_1}}$ and store K_{s_1} for future use.

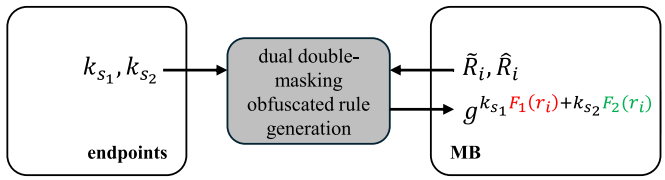
C. Preprocessing

This phase is only executed in the first session. In this phase, **MB** runs a preprocessing protocol with the endpoints to generate the obfuscated rules. The preprocessing protocol is given in Fig. 3. Specifically, each endpoint first sends its K_{s_1} to **MB**. Upon receiving K_{s_1} from each endpoint, **MB** checks whether the received K_{s_1} are the same. If yes, it sends $(R, Sig_{RG}(R), Sig_{MB}(R), \{\widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), Sig_{MB}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]})$ to the endpoints, where $R = g^{absr}$, $\widetilde{R}_i = g^{yabsrH_2(r_i)+\tilde{y}H_3(g^{absrH_2(r_i)})}$, $\widehat{R}_i = g^{yH_3(g^{absrH_2(r_i)})}$. Each endpoint checks whether the signatures are valid to make sure the received $(R, \{\widetilde{R}_i, \widehat{R}_i\}_{i\in[n]})$ are the right parameters generated by the rule preparation protocol. If yes, each endpoint calculates $\widetilde{K}_i = (\widetilde{R}_i)^{k_{s_1}} \cdot (\widehat{R}_i)^{k_{s_2}}$ for $i \in [n]$, returns $\{\widetilde{K}_i\}_{i\in[n]}$ to **MB**, and stores R that will be used to generate the encrypted token during the subsequent token encryption phase. **MB** then checks if the received $\{\widetilde{K}_i\}_{i\in[n]}$ are the same. If yes, it calculates $K_i = (\widetilde{K}_i/(K_{s_1})^{y\tilde{y}H_3(R_i)})^{1/y} = g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})}$ for $i \in [n]$.

The rationale of this protocol is described as follows. To enable **MB** to inspect the encrypted traffic, **MB** needs to obtain the output of a function $F(x_1, x_2)$, where the input x_1 is derived from the secret owned by the endpoint(s) and the input x_2 is derived from a rule r_i for $i \in [n]$. To that end, **MB** runs the preprocessing protocol with the endpoints to derive $F(x_1, x_2)$.



(a) Rule preparation protocol



(b) Preprocessing protocol

Fig. 4. A diagram of dual double-masking obfuscated rule generation. The output of **MB** is in the form of $g^{aF_1(r_i)+bF_2(r_i)}$ where $F_1(r_i) = absrH_2(r_i)$ and $F_2(r_i) = H_3(g^{absrH_2(r_i)})$. We call this form as dual double-masking structure.

In the protocol (as described in Fig. 3), the input x_1 is (k_{s_1}, k_{s_2}), the input x_2 is the rule tuple (generated by the rule preparation protocol) of r_i for $i \in [n]$, and the output of $F(x_1, x_2)$ is $K_i = g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})}$ for $i \in [n]$. The security requirements here include (1) the endpoint(s) should not obtain r_i while **MB** cannot know (k_{s_1}, k_{s_2}); and (2) anyone (including **MB** or **RG** alone) cannot forge a valid obfuscated rule for $r^* \notin \{r_i\}_{i\in[n]}$. The rule preparation protocol in Fig. 2 and the preprocessing protocol given in Fig. 3 are designed (together) to satisfy such requirements.

D. Session Rule Preparation

The underlying design of these two protocols is a new technique that we introduced, called *dual double-masking obfuscated rule generation*, see Fig. 4. In particular, the rule tuple of rule r_i (generated by the rule preparation

protocol in Fig. 2) is in the form of $(\widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), Sig_{MB}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i), Sig_{MB}(\widehat{R}_i))$, where $\widetilde{R}_i = g^{yabsrH_2(r_i)+y\widetilde{y}H_3(g^{absrH_2(r_i)})}$, $\widehat{R}_i = g^{yH_3(g^{absrH_2(r_i)})}$. The exponent of $\widetilde{R}_i$ consists of two parts: $yabsrH_2(r_i)$ and $y\widetilde{y}H_3(g^{absrH_2(r_i)})$, each of which is generated from the secret of **MB** (i.e., y and $\widetilde{y}$) and the hash value of r_i (i.e., $H_2(r_i)$ and $H_3(g^{absrH_2(r_i)})$). In other words, $H_2(r_i)$ and $H_3(g^{absrH_2(r_i)})$ are masked by y and $\widetilde{y}$. The obfuscated rule K_i of r_i (generated by the preprocessing protocol in Fig. 3) is in the form of $g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})}$. Similarly, the exponent of K_i consists of the following two parts: $k_{s_1}absrH_2(r_i)$ and $k_{s_2}H_3(g^{absrH_2(r_i)})$, each of which is generated from the secret of the endpoint (i.e., k_{s_1} and k_{s_2}) and the hash value of r_i (i.e., $H_2(r_i)$ and $H_3(g^{absrH_2(r_i)})$). That is, $H_2(r_i)$ and $H_3(g^{absrH_2(r_i)})$ are masked by k_{s_1} and k_{s_2}. We call such masking structure as the *dual double-masking structure*.

In particular, $\widetilde{R}_i$ and $\widehat{R}_i$ are elaborately designed such that (1) enable **RG** and **MB** to run the preprocessing protocol in this phase to generate the obfuscated rule of r_i; (2) do not leak r_i to the endpoints; and (3) do not enable anyone (including **MB** or **RG** alone) to forge the valid obfuscated rule for any $r^* \notin \{r_i\}_{i\in[n]}$.

In the preprocessing protocol, each endpoint is given $R = g^{absr}$, $\{\widetilde{R}_i = g^{yabsrH_2(r_i)+y\widetilde{y}H_3(g^{absrH_2(r_i)})}, \widehat{R}_i = g^{yH_3(g^{absrH_2(r_i)})}\}_{i\in[n]}$. Let $b_1 = absr$, $b_{2,i} = H_2(r_i)$ and $b_{3,i} = H_3(g^{absrH_2(r_i)})$, then the exponents of the messages the endpoint received (i.e., R, $\widetilde{R}_i$ and $\widehat{R}_i$) are $(b_1, \{yb_1b_{2,i} + y\widetilde{y}b_{3,i}, yb_{3,i}\}_{i\in[n]})$. Clearly, since y and $\widetilde{y}$ are unknown to the endpoint, b_1, $\{yb_1b_{2,i} + y\widetilde{y}b_{3,i}\}_{i\in[n]}$ and $\{yb_{3,i}\}_{i\in[n]}$ are random from the endpoint's point of view. This ensures that the endpoint cannot obtain r_i. In terms of **MB** (resp. **RG**), the messages it receives from the network are K_{s_1} and $\{\widetilde{K}_i\}_{i\in[n]}$, and the corresponding exponents are k_{s_1} and $\{k_{s_1}(yb_1b_{2,i} + y\widetilde{y}b_{3,i}) + k_{s_2}yb_{3,i}\}_{i\in[n]}$. Recall that k_{s_1} and k_{s_2} are unknown to **MB** (resp. **RG**), we have that k_{s_1} and $\{k_{s_1}(yb_1b_{2,i} + y\widetilde{y}b_{3,i}) + k_{s_2}yb_{3,i}\}_{i\in[n]}$ are random because the determinants of the following coefficient matrices are all nonzero:

$$\begin{vmatrix} 1 & 0 \\ yb_1b_{2,i} + y\widetilde{y}b_{3,i} & yb_{3,i} \end{vmatrix} \neq 0, \quad \begin{vmatrix} yb_1b_{2,i'} + y\widetilde{y}b_{3,i'} & yb_{3,i'} \\ yb_1b_{2,j'} + y\widetilde{y}b_{3,j'} & yb_{3,j'} \end{vmatrix} \neq 0,$$

for $i \in [n]$, $i', j' \in [n]$ s.t. $i' \neq j'$. This ensures that **MB** (resp. **RG**) cannot forge a valid obfuscated rule.

A session rule preparation protocol will be executed in this phase. Let k' be the session key of a new session, and $k'_{SSL}, k'_s \in \mathbb{Z}_p$ be the keys derived from k' by the endpoints, where k'_{SSL} is the regular SSL key of the new session and k'_s is the key used to generate session rules of the new session in this phase. The session rule preparation protocol is described in Fig. 5, which aims to generate session rule I_i of rule r_i for the current session from the obfuscated rule K_i.

Specifically, for rule r_i, since K_i is generated from the secret (k_{s_1}, k_{s_2}) of the endpoints in the first session, K_i can be directly set as the session key for the first session. For any subsequent session different from the first session, each endpoint will send $K_s = g^{k'_s}$ to **MB**. Since at least one of the endpoints is assumed to be honest, to check whether the received K_s is the correct one, **MB** can simply check whether the received values are the same. If so, **MB** calculates $I_i = K_i \cdot K_s$, which is used to adapt K_i to the new session.

Input: The endpoints have common inputs k'_s; **MB** has inputs $(\{K_i\}_{i\in[n]})$, where $K_i = g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})}$.

Protocol:

- If it is the first session, **MB** sets $I_i = K_i$ for $i \in [n]$.
- If it is a subsequent session:
  1. Each endpoint sends $K_s = g^{k'_s}$ to **MB**.
  2. **MB** checks whether the received K_s are the same. If yes, it calculates $I_i = K_i \cdot K_s$ for $i \in [n]$; otherwise, it halts and aborts.

Fig. 5. Session rule preparation protocol.

As noted in [27], more and more group elements will be consumed as the number of session increases. The protocol in Fig. 3 will be called every N sessions (where N depends on the group we chosen), which is used to prevent potential brute-force attack.

E. Token Encryption

We utilize the window-based tokenization methodology. As described in BlindBox, the windows-based tokenization extracts tokens from a given packet by utilizing the sliding window algorithm. We parse a bytestream by a fixed length of 8 bytes. For a traffic "keep secret" consisting of 11 bytes of character, the tokens derived by the windows-based tokenization are "keep sec", "eep secr", "ep secre" and "p secret". To reduce the bandwidth overhead, we can further utilize the delimiter-based tokenization as introduced in BlindBox. Delimiters are spacing, punctuation, and special symbols. For a payload "login.jsp?username=carl", the possible keywords in rules are "login", "login.jsp", "?username", "username=carl", instead of "ogin" or "ogin.j". In this sense, the endpoint(s) only needs to generate tokens that could match rules that start and end on delimiter-based offsets. In this way, the redundant tokens in the window can be ignored.

In this phase, for each token t_i, **S** runs the token encryption algorithm described in Fig. 6. To reuse the session token T_i of token t_i that has been generated, **S** first initializes a counter table CT that stores a tuple $(\mathsf{count}_{t_i}, T_{t_i}, t_i)$ for each token t_i, where count_{t_i} is the times t_i appeared in the packet. Recall that the session rule I_i of rule r_i is in the form of $g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})}$ for the first session (resp. $g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})} \cdot g^{k'_s}$ for a subsequent session). To enable **MB** to detect the encrypted traffic during the traffic inspection phase, the session token of a token t_i (used to generate the corresponding encrypted token) should be in the form of $g^{k_{s_1}absrH_2(t_i)+k_{s_2}H_3(g^{absrH_2(t_i)})}$ for the first session (resp. $g^{k_{s_1}absrH_2(t_i)+k_{s_2}H_3(g^{absrH_2(t_i)})} \cdot g^{k'_s}$ for a subsequent session). The token encryption algorithm in Fig. 6 is designed to satisfy the above requirement. To prevent frequency

Input: **S** has inputs R, a token t_i, CT, k_{s_1}, k_{s_2}, S_{salt} and K_s (for a subsequent session), where $R = g^{absr}$, $K_s = g^{k'_s}$.

Algorithm: If it is the first session:

- If t_i does not appear in any tuple in CT: set $\mathsf{count}_{t_i} := 0$, calculate $\widetilde{T}_{t_i} = (R)^{H_2(t_i)}$, $T_{t_i} = (\widetilde{T}_{t_i})^{k_{s_1}} \cdot g^{k_{s_2} H_3(\widetilde{T}_{t_i})}$ and $D_{t_i} = H_4(S_{salt}, T_{t_i})$, and insert $(\mathsf{count}_{t_i}, T_{t_i}, t_i)$ into CT.
- If there exists a tuple $(\mathsf{count}_{t'_i}, T_{t'_i}, t'_i)$ in CT where $t'_i = t_i$: update $\mathsf{count}_{t'_i} := \mathsf{count}_{t'_i} + 1$ and calculate $D_{t_i} = H_4(S_{salt} + \mathsf{count}_{t'_i}, T_{t'_i})$.

If it is a subsequent session: Calculate $\widetilde{T}_{t_i} = (R)^{H_2(t_i)}$, $T_{t_i} = (\widetilde{T}_{t_i})^{k_{s_1}} \cdot g^{k_{s_2} H_3(\widetilde{T}_{t_i})} \cdot K_s$,

- If t_i does not appear in any tuple in CT: set $\mathsf{count}_{t_i} := 0$, calculate $D_{t_i} = H_4(S_{salt}, T_{t_i})$, and insert $(\mathsf{count}_{t_i}, T_{t_i}, t_i)$ into CT.
- If there exists a tuple $(\mathsf{count}_{t'_i}, T_{t'_i}, t'_i)$ in CT where $t'_i = t_i$ and $T_{t'_i} \neq T_{t_i}$: set $\mathsf{count}_{t'_i} := 0$, $T_{t'_i} := T_{t_i}$, and calculate $D_{t_i} = H_4(S_{salt}, T_{t'_i})$.
- If there exists a tuple $(\mathsf{count}_{t'_i}, T_{t'_i}, t'_i)$ in CT where $t'_i = t_i$ and $T_{t'_i} = T_{t_i}$: update $\mathsf{count}_{t'_i} := \mathsf{count}_{t'_i} + 1$ and calculate $D_{t_i} = H_4(S_{salt} + \mathsf{count}_{t'_i}, T_{t'_i})$.

Fig. 6. Token encryption algorithm.

analysis attack, we insert a random salt during the encryption of each token, as described in Fig. 6. The initial salt S_{salt} is derived from k_r by **S**, which will be sent to **MB**.[4] There are two cases for the token encryption. If it is the first session, the session rule T_{t_i} is set to be $(R)^{k_{s_1} H_2(t_i)} \cdot g^{k_{s_2} H_3(\widetilde{T}_{t_i})}$; otherwise (i.e., it is a subsequent session), T_{t_i} is set to be $(R)^{k_{s_1} H_2(t_i)} \cdot g^{k_{s_2} H_3(\widetilde{T}_{t_i})} \cdot g^{k'_s}$, where $\widetilde{T}_{t_i} = (R)^{H_2(t_i)}$. For a given token t_i, if it does not appear in any tuple in CT, the encrypted token D_{t_i} is generated as $H_4(S_{salt}, T_{t_i})$; otherwise, D_{t_i} is generated as $H_4(S_{salt} + \mathsf{count}_{t_i}, T_{t_i})$, where count_{t_i} is the frequency t_i appeared in the stream so far. To prevent the count table from growing too large,[5] **S** can reset CT, generate a new salt $\widetilde{S_{salt}} = S_{salt} + \max_{t_i} \mathsf{count}_{t_i} + 1$ and send it to **MB**.

F. Traffic Inspection

Upon receiving the encrypted tokens sent from **S**, **MB** performs the traffic inspection algorithm, which is described in Fig. 7. Similar to PrivDPI, **MB** employs a fast search tree that contains E_{r_i} and a count table CT_m that includes tuple $(\mathsf{count}_{r_i}, E_{r_i})$ for each rule r_i, where count_{r_i} is the count of rule r_i (initialized to be 0) and E_{r_i} is the encrypted rule of r_i corresponding to count_{r_i}. During this phase, an equality check between encrypted rules and encrypted tokens is performed by **MB**. To this end, given an encrypted token $D_{t_i} = H_4(S_{salt} + \mathsf{count}_{t_i}, T_{t_i})$, **MB** only needs to calculate the encrypted rule $E_{r_i} = H_4(S_{salt} + \mathsf{count}_{r_i}, I_i)$ and checks if E_{r_i} equals D_{t_i}.

[4]To ensure whether the received S_{salt} is correct, **MB** may ask each endpoint to send its S_{salt} and check whether the received value are the same.

[5]Note search overhead will increase as the growth of the count table.

Input: **MB** has inputs S_{salt}, a fast search tree, a count table CT_m and $\{I_i\}_{i\in[n]}$.

Algorithm: For each D_{t_i}, **MB** tests if there exists an $E_{r_{i'}}$ equals D_{t_i} for some $i' \in [n]$, if yes, operate as follows:

1. Take the corresponding action.
2. Delete the node in tree corresponding to rule $r_{i'}$, and insert $E_{r_{i'}} = H_4(S_{salt} + \mathsf{count}_{r_{i'}} + 1, I_{i'})$.
3. Set $\mathsf{count}_{r_{i'}} := \mathsf{count}_{r_{i'}} + 1$.

Fig. 7. Traffic inspection algorithm.

G. Traffic Validation

The aim of this phase is to prevent **S** from encrypting a different, benign payload during the generation of encrypted traffic to evade inspection. Specifically, upon receiving the encrypted traffic forwarded by **MB**, **R** will run this traffic validation algorithm to check whether the encrypted traffic is honestly encrypted from the payload. To this end, **R** takes the decrypted traffic from the TLS encrypted traffic and encrypts the decrypted traffic using the same token encryption algorithm as used by **S**. **R** then checks whether the resulting ciphertext is the same as the encrypted traffic received. If not, it concludes that **S** is malicious and aborts.

V. Variant of PrivBox

A. Enhanced Rule Preparation Protocol

The rule preparation protocol in Fig. 2 requires the pairing operation to check whether the parameters are correctly generated using **RG**'s secret r. Generally, the elliptic curves supporting pairing are less efficient than other normal elliptic curves. Intuitively, we can select a more efficient elliptic curve to reduce the computation overhead of the rule preparation protocol. To this end, we utilize the technique of non-interactive (one-round) zero-knowledge proof of knowledge (via applying the Fiat-Shamir heuristic [16]) to eliminate the usage of pairing operation. In the following, we take the checking of $\widehat{R}$ as an example to illustrate how to replace the pairing operation with non-interactive (one-round) zero-knowledge proof of knowledge. Recall that to check whether $\widehat{R}$ is correctly generated from $\widetilde{R}$ and **RG**'s secret r, the rule preparation protocol in Fig. 2 needs to check whether $e(\widetilde{R}, g) \stackrel{?}{=} e(\widehat{S}_A, L)$ holds, which requires two pairings. We observe that such checking is the same to the problem of proving knowledge of a discrete logarithm (i.e., **RG**'s secret r). A promising technique for proving knowledge of a discrete logarithm is the technique of the well-known Schnorr protocol [30], a three-round example of a zero-knowledge protocol. To prove the knowledge of r for $\widehat{R}$, **RG** first randomly chooses $x_1 \in \mathbb{Z}_p$, calculates $Z_1 = (\widetilde{R})^{x_1}$, and sends Z_1 to **MB**. Upon receiving Z_1, **MB** randomly chooses $c_1 \in \mathbb{Z}_p$ as its challenge and returns c_1 to **RG**. **RG** then calculates $z_1 = rc_1 + x_1$ and sends z_1 to **MB**. Once receiving z_1, **MB** can conclude that $\widehat{R}$ is correctly generated from $\widetilde{R}$ and **RG**'s secret r if $(\widetilde{R})^{z_1} \stackrel{?}{=} Z_1 \cdot (R)^{c_1}$ holds. However, as shown above, simply

Input: **RG** has inputs $(\{r_i\}_{i\in[n]}, a, r, H_1, H_2)$ and the public/private key pair $(pk_{Sig_{RG}}, sk_{Sig_{RG}})$ of Sig_{RG}. **MB** has inputs $(\{r_i\}_{i\in[n]}, b, s, y, \widetilde{y})$ and the public/private key pair $(pk_{Sig_{MB}}, sk_{Sig_{MB}})$ of Sig_{MB}.

Protocol:

1. **RG** computes $S_A = g^a$, $L = g^r$, and sends them to **MB**.
2. **MB** calculates $\widetilde{S}_B = g^b$, $S = g^s$, and sends (S_B, S) to **RG**.
3. **RG** calculates $\widetilde{S}_A = (S_B)^a = g^{ab}$, $V_i = (\widetilde{S}_A)^{H_2(r_i)} = g^{abH_2(r_i)}$ for $i \in [n]$, and sends $(\{V_i\}_{i\in[n]})$ to **MB**.
4. **MB** calculates $\widetilde{S}_B = (S_A)^b$ and checks whether the equation $V_i \stackrel{?}{=} (\widetilde{S}_B)^{H_2(r_i)}$ holds for $i \in [n]$. If not, it halts and aborts. Otherwise, it calculates $Y = \mathsf{Enc}_{H_1(\widetilde{S}_B)}(y||\widetilde{y})$, $\widetilde{R} = (\widetilde{S}_B)^s$, $S_i = V_i^s$ for $i \in [n]$, and sends $(Y, \widetilde{R}, \{S_i\}_{i\in[n]})$ to **RG**.
5. **RG** runs $\mathsf{Dec}_{H_1(\widetilde{S}_A)}(Y)$ to obtain $(y, \widetilde{y})$, chooses random $x_1, x_2 \in \mathbb{Z}_p$ and computes $R = (\widetilde{R})^r$, $\widehat{R} = R \cdot \widetilde{S}_A$, $R_i = (S_i)^r$, $\widetilde{R}_i = (R_i)^y \cdot g^{y\widetilde{y}\cdot H_3(R_i)}$, $\widehat{R}_i = g^{y\cdot H_3(R_i)}$ for $i \in [n]$, $Z_1 = (\widetilde{R})^{x_1}$, $Z_2 = (g \cdot \prod_{i=1}^n S_i)^{x_2}$, $c_1 = H_3(Z_1)$, $c_2 = H_3(Z_2)$, $z_1 = rc_1 + x_1$, $z_2 = rc_2 + x_2$, signs R, $\widetilde{R}_i$, $\widehat{R}_i$ with $sk_{Sig_{RG}}$ to obtain the signatures $Sig_{RG}(R), \{Sig_{RG}(\widetilde{R}_i), Sig_{RG}(\widehat{R}_i)\}_{i\in[n]}$. It then sends $(Z_1, z_1, Z_2, z_2, \widehat{R}, Sig_{RG}(R), \{R_i, \widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i)\}_{i\in[n]})$ to **MB**, where $R = g^{absr}$, $R_i = g^{absrH_2(r_i)}$, $\widetilde{R}_i = g^{yabsrH_2(r_i)+y\widetilde{y}H_3(g^{absrH_2(r_i)})}$, $\widehat{R}_i = g^{yH_3(g^{absrH_2(r_i)})}$.
6. For $i \in [n]$, **MB** computes $c_1 = H_3(Z_1)$, $c_2 = H_3(Z_2)$, $R = \widehat{R}/\widetilde{S}_B$, and checks whether
   1) $Sig_{RG}(R)$ is a valid signature of R, $Sig_{RG}(\widetilde{R}_i)$ is a valid signature of $\widetilde{R}_i$ and $Sig_{RG}(\widehat{R}_i)$ is a valid signature of $\widehat{R}_i$ for $i \in [n]$;
   2) $(\widetilde{R})^{z_1} \stackrel{?}{=} Z_1(R)^{c_1}$ holds;
   3) $(g \cdot \prod_{i=1}^n S_i)^{z_2} \stackrel{?}{=} Z_2(L \cdot \prod_{i=1}^n R_i)^{c_2}$, $\widetilde{R}_i \stackrel{?}{=} (R_i)^y \cdot g^{y\widetilde{y}H_3(R_i)}$, $\widehat{R}_i \stackrel{?}{=} g^{yH_3(R_i)}$ hold for $i \in [n]$.

   If not, it halts and aborts. Otherwise, it signs R, $\{\widetilde{R}_i, \widehat{R}_i\}_{i\in[n]}$ with $sk_{Sig_{MB}}$ to obtain the signatures $Sig_{MB}(R)$, $\{Sig_{MB}(\widetilde{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]}$, and finally stores $(y, \widetilde{y}, R, Sig_{RG}(R), Sig_{MB}(R), \{\widetilde{R}_i, Sig_{RG}(\widetilde{R}_i), Sig_{MB}(\widetilde{R}_i), \widehat{R}_i, Sig_{RG}(\widehat{R}_i), Sig_{MB}(\widehat{R}_i)\}_{i\in[n]})$ for future use.

Fig. 8. Enhanced rule preparation protocol.

applying technique of the Schnorr protocol incurs two additional rounds. Fortunately, the Fiat-Shamir heuristic allows us to transform the three-round Schnorr protocol into non-interactive (one-round) zero-knowledge proof of knowledge. In particular, we can modify the above three-round protocol as follows: **RG** first randomly chooses $x_1 \in \mathbb{Z}_p$, calculates $Z_1 = (\widetilde{S})^{x_1}$, $c_1 = H_3(Z_1)$, $z_1 = rc_1 + x_1$, and sends (z_1, Z_1) to **MB**. Upon receiving (z_1, Z_1), **MB** computes $c_1 = H_3(Z_1)$ and accepts if $(\widetilde{R})^{z_1} = Z_1 \cdot (R)^{c_1}$. Similarly, we can apply the same idea to check whether $\{R_i\}_{i\in[n]}$ are correctly generated. The enhanced rule preparation protocol is described in Fig. 8.

B. Supporting Full IDS Functionality

The system described in Section IV provides the guarantee of exact match privacy, i.e., **MB** could learn the position of an attack keyword occurs in a flow. In this section, we extend PrivBox to support full IDS functionality (i.e., the guarantee of probable-cause privacy model) such that **MB** is able to decrypt the encrypted traffic if a token is matched, allowing **MB** to inspect (e.g., run regular expression) on the plaintext. The idea is to attach the regular SSL key k_{SSL} to the encrypted token such that **MB** will obtain k_{SSL} if the token is matched. In particular, for a token t, we add $D'_{t_i} = H_4(S_{salt} + \mathsf{count}_{t_i}, T_{t_i}) \oplus k_{SSL}$ as the additional part of the encryption of token t, where $T_{t_i} = g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})}$ for the first session (resp. $T_{t_i} = g^{k_{s_1}absrH_2(r_i)+k_{s_2}H_3(g^{absrH_2(r_i)})} \cdot g^{k'_s}$ for a subsequent session). Since **MB** knows $E_r = H_4(S_{salt} + \mathsf{count}_r, I)$ for a rule r where $I = g^{k_{s_1}absrH_2(r)+k_{s_2}H_3(g^{absrH_2(r)})}$ for the first session (resp. $I = g^{k_{s_1}absrH_2(r)+k_{s_2}H_3(g^{absrH_2(r)})} \cdot g^{k'_s}$ for a subsequent session), if t_i is matched by a rule r, it can obtain k_{SSL} by calculating $D'_{t_i} \oplus E_r$.

VI. Security

A. Rule Preparation Security

From the security point of view, the rule preparation protocol is a two-party computation that maps pairs of inputs to pairs of outputs. By f_{RP} we denote the process of the computation, which is defined as $f_{RP} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$. Let a_i be the i-th party's input, and $(f^1_{RP}(a_1, a_2), f^2_{RP}(a_1, a_2))$ be the output pair. For the rule preparation protocol in Section IV-B, the two parties are **MB** and **RG**. For a clear description of our proof, we give a simplified version of rule preparation protocol in Appendix B, available online, and the security definition of the rule preparation protocol is given in Appendix C, available online.

Theorem 1: Assuming that the data encapsulation mechanism Π is CPA-secure and computing discrete logarithms is hard in the group G, the rule preparation protocol securely computes the functionality f_{RP}.

Proof: The proof of this theorem is given in Appendix D, available online. □

B. Preprocessing Security

The preprocessing protocol in Section IV-C is also a two-party computation. By f_{PP} we denote the process of the computation, which is defined as $f_{PP} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$. Let b_i be the input of the i-th party, for every input

pair (b_1, b_2), the output pair is $(f_{PP}^1(b_1, b_2), f_{PP}^2(b_1, b_2))$. For the preprocessing protocol, the two parties are **MB** and the endpoint(s). For a clear description of our proof, we give a simplified version of preprocessing protocol in Appendix E, available online, and the security definitions, including confidentiality and non-malleability [7], [17] are given in Appendix F, available online.

Theorem 2: If computing discrete logarithms is hard in the group G, the preprocessing protocol securely computes the function f_{PP}.

Proof: The proof of this theorem is given in Appendix G, available online. □

Theorem 3: If computing discrete logarithms is hard in the group G, the preprocessing protocol is non-malleable.

Proof: The proof of this theorem is given in Appendix H, available online. □

C. Middlebox Searchable Encryption

The underlying key component of PrivBox is a notion called *middlebox searchable encryption scheme* (MBSE). MBSE provides a matching mechanism for middlebox-based encrypted traffic inspection. An MBSE scheme with message space $\mathcal{M}$ is defined as follows.

- $\mathsf{Setup}(1^\lambda)$: This algorithm takes as input the security parameter 1^λ, and outputs the public parameters pp and a key k.
- $\mathsf{RuleEnc}(pp, k, r)$: The rule encryption algorithm takes as input the public parameters pp, the key sk, and a rule $r \in \mathsf{R}$. It outputs an encrypted rule ER of r.[6]
- $\mathsf{TokenEnc}(pp, k, \{t_i\}_{i\in[d]})$: The token encryption algorithm takes as input the public parameters pp, the key k, and a token set $\{t_i\}_{i\in[d]}$, where $t_i \in \mathcal{M}$ for $i \in [d]$. It outputs a salt S_{salt} and a set of ciphertexts $\{CT_i\}_{i\in[d]}$.
- $\mathsf{Match}(ER, S_{salt}, \{CT_i\}_{i\in[d]})$: The match algorithm takes as input ER, a salt S_{salt} and $\{CT_i\}_{i\in[d]}$. It outputs an index set $\mathcal{S} = \{s_1, \ldots, s_m\}$, where $s_i \in [d]$ for $i \in [m]$.

The correctness and security of MBSE are given in Appendix I and Appendix J, available online, respectively. For a clear description of our proof, we present an MBSE scheme that outlines the main structure of PrivBox (in the first session) from the security point of view, which is given in Appendix K, available online.

Theorem 4: If computing discrete logarithms is hard in the group G and H_4 is a random oracle, the MBSE scheme presented in Appendix K, available online, is secure under the definition given in Appendix J, available online.

Proof: The proof of this theorem is given in Appendix L, available online. □

VII. Performance Evaluation

We evaluated the performance of PrivBox by implementing the main construction in Section IV with the variant described in Section V using the `Charm-Crypto` framework. Our experiment was conducted on the prime256v1 curve (known as NIST Curve P-256), and we utilized `pyOpenSSL` library for establishing a TLS connection between the endpoints. Besides, we used the Linux Traffic Control (`tc`) tool to emulate a realistic network environment and the traffic between endpoints was throttled to 100 Mbps to align with the modern WAN network. Our middlebox and prototype of the client software run with an Intel(R) Xeon(R) E5-2680 CPU with 4 cores at 2.40 GHz under 64-bit Linux OS (Ubuntu 18.04). The CPU supports AES-NI instructions. We implemented the hash function H with AES-256 algorithm.

Since PrivBox follows the setting of BlindBox, PrivDPI and P2DPI, and PrivDPI, P2DPI outperform BlindBox, we compared PrivBox with PrivDPI and P2DPI in the experiments. We also compared PrivBox with a recent DPI system Pine [26] for completeness. We used ECDSA as the signature deployed in both PrivDPI and PrivBox. To conduct a comprehensive comparison, we used the rulesets from Snort [34], adapting to different cases in our experiments, in which a rule is tokenized to a size of 8 bytes. We repeated each instance of our experiments 10,000 times and eventually took the average.

TABLE I
Computational Complexity of Different Schemes*

Algorithms	Computational complexity			
	PrivDPI	Pine	P2DPI	PrivBox
Setup	**RG**: $\mathcal{O}(n)$	**MB**: $\mathcal{O}(n)$ **RG**: $\mathcal{O}(n)$	**RG**: $\mathcal{O}(n)$	**RG**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$
Preprocessing	**S**: $\mathcal{O}(n)$ **R**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$	**S**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$
Session rule preparation	**S/R**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$ **RG**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$
Token encryption	**S**: $\mathcal{O}(m)$	**S**: $\mathcal{O}(m)$	**S**: $\mathcal{O}(m)$	**S**: $\mathcal{O}(m)$
Traffic inspection	**MB**: $\mathcal{O}(mn)$	**MB**: $\mathcal{O}(mn)$	**MB**: $\mathcal{O}(mn)$	**MB**: $\mathcal{O}(mn)$

*: In this table, we denote m and n as the number of tokens and rules, respectively.

A. Complexity Analysis

We first compare the computational and space complexity of different schemes in theory with results reported in Tables I and II. From Table I, we observe that all four schemes have a $\mathcal{O}(n)$ complexity for setup in **RG**, while PrivBox and Pine require additional operations after **MB** receives the rules from **RG**, such as validating the signature of each rule. During preprocessing, **MB** generates obfuscated rules with endpoints, but the way endpoints (**S** and **R**) interact varies among schemes. In P2DPI and PrivBox, both **S** and **R** must verify rules prepared by **MB** to mitigate malicious behavior. Instead, PrivDPI and Pine only require verification on **S**, with PrivDPI additionally performing a lightweight parameter initialization on **R**. Unlike the other three schemes, P2DPI does not support reusable obfuscated rules, necessitating repeated setup and preprocessing during session rule preparation (see Section VII-B3 for a concrete overhead comparison). The other schemes do not require significant endpoint intervention in this phase. Finally, all four schemes

[6]Here, by encrypted rule ER we mean the session rule of the first session as described in Section IV-D, which outlines the main structure of PrivBox from the security perspective.

TABLE II
SPACE COMPLEXITY OF DIFFERENT SCHEMES*

Algorithms	Space complexity			
	PrivDPI	Pine	P2DPI	PrivBox
Setup	**RG**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$	**S**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$ **RG**: $\mathcal{O}(n)$	**RG**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$	**RG**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$
Preprocessing	**S**: $\mathcal{O}(n)$ **R**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$	**S**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$
Session rule preparation	**S/R**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(n)$ **MB**: $\mathcal{O}(n)$ **RG**: $\mathcal{O}(n)$	**S/R**: $\mathcal{O}(1)$ **MB**: $\mathcal{O}(n)$
Token encryption	**S**: $\mathcal{O}(m)$	**S**: $\mathcal{O}(m)$	**S**: $\mathcal{O}(m)$	**S**: $\mathcal{O}(m)$
Traffic inspection	**MB**: $\mathcal{O}(m+n)$	**MB**: $\mathcal{O}(m+n)$	**MB**: $\mathcal{O}(m+n)$	**MB**: $\mathcal{O}(m+n)$

*: In this table, we denote m and n as the number of tokens and rules, respectively.

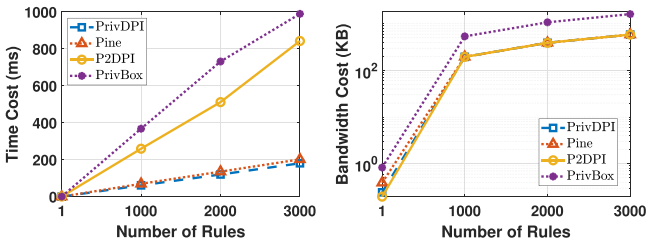


Fig. 9. Time and bandwidth cost of **MB** in the preprocessing phase (in the first session).

have the same complexity in token encryption ($\mathcal{O}(m)$, handling tokens sequentially) and traffic inspection ($\mathcal{O}(mn)$, detecting each token across all given rules).

From Table II, we observe that the space complexity for setup, preprocessing, session rule preparation, and token encryption closely aligns with their computational complexity. This is well understood as computation over a linear (or constant) number of inputs requires corresponding linear (or constant) storage to record results. The only difference lies in setup, where, after receiving detection rules from **RG**, PrivDPI and P2DPI require $\mathcal{O}(n)$ additional space for storing them locally in **MB**. Since **MB** in four schemes employs a similar detection approach, utilizing a tree-based structure and a token list as inputs, all traffic inspection phases have a space complexity of $\mathcal{O}(m+n)$.

B. Performance of Middlebox

For a connection between **S** and **R**, the main computation and communication overheads are from the preprocessing, the session rule preparation and the traffic inspection phases.

1) Performance of the Preprocessing Phase in the First Session: Time Cost: The preprocessing time in the first session of **MB** is shown in Fig. 9(a). Though PrivBox takes more time than other systems in the first session, the overhead is still relatively small (especially for the fact that **MB** is generally deployed in a powerful cloud server).

Bandwidth Cost: As shown in Fig. 9(b), PrivBox incurs more bandwidth than other systems, however, it is still acceptable.

TABLE III
TIME OF **MB** IN THE TRAFFIC INSPECTION PHASE

No. of Rules	Time (μs)			
	PrivDPI	Pine	P2DPI	PrivBox
1	0.477	0.477	75.006	0.477
1,000	113.892	113.63	12,432.098	113.706
2,000	230.527	230.241	25,375.175	230.981
3,000	340.655	340.652	38,547.993	340.711

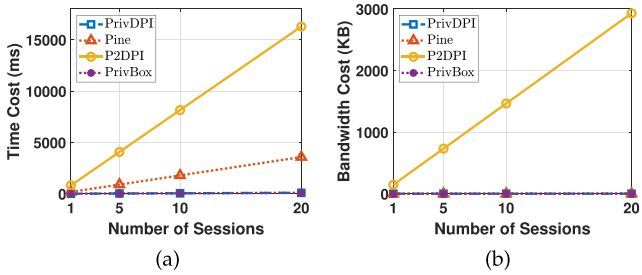


Fig. 10. Time and bandwidth cost of **MB** in the preparation of session rule (in the subsequent session).

The reason is that all of these systems only send a few group elements for each rule.

2) Performance of Traffic Inspection Phase in the First Session: Since there is no communication cost in this phase, we only consider the traffic inspection time. Table III shows the one token inspection time for four different rulesets, which indicates that PrivBox, PrivDPI and Pine share similar traffic inspection delay, which is quite low. In particular, PrivBox only takes 340.711 μs for inspecting one token with a ruleset of 3,000 rules.

3) Performance of the Preparation of Session Rule in Subsequent Session: To generate a session rule, PrivDPI, Pine and PrivBox need to run the preprocessing protocol and the session rule preparation protocol for the first session. For any subsequent session, they just run the session rule preparation protocol for the same rule. For P2DPI, it needs to run the *rule setup* protocol for every session.

Time Cost: For a ruleset of 3,000, the time costs for 1, 5, 10 and 20 consecutive sessions are given in Fig. 10(a). It shows that the time cost of PrivBox is similar to that of PrivDPI. In particular, for 10 consecutive sessions, PrivBox only takes 45.605 ms, which is over 38x faster than Pine and over 177x faster than P2DPI. This is due to the fact that Pine needs to run an extra *session detection rule preparation* protocol for every session, and P2DPI needs to run the rule setup protocol for every session.

Bandwidth Cost: As shown in Fig. 10(b), the bandwidth incurred by PrivBox is relatively low (only 0.488 KB for 10 consecutive sessions), which is the same as PrivDPI and Pine. This is due to the fact that PrivBox, PrivDPI and Pine only need to send a group element for the preparation of session rule. In contrast, P2DPI needs to recompute and send session rules for every session, as a result, it incurs relatively high bandwidth as compared to other systems.

TABLE IV
TIME OF ENDPOINT IN THE PREPROCESSING PHASE (IN THE FIRST SESSION)

No. of Rules	Time (ms)			
	PrivDPI	Pine	P2DPI	PrivBox
1	0.438	0.463	0.482	0.689
1,000	238.399	244.046	248.626	296.511
2,000	480.222	475.962	481.251	597.934
3,000	753.599	735.245	703.125	890.42

TABLE V
TIME OF ENDPOINT IN TOKEN ENCRYPTION

No. of Tokens	Time (ms)			
	PrivDPI	Pine	P2DPI	PrivBox
1	0.169	0.158	0.081	0.214
100	8.302	8.742	7.476	20.993
500	41.278	44.199	37.716	101.86
800	66.173	70.699	59.718	163.313

4) Performance of Traffic Inspection Phase in a Subsequent Session: As described before (cf. Table III), the traffic inspection delay in PrivBox is close to that of PrivDPI and Pine. For a ruleset of 3,000, the gap is <0.1 μs.

C. Performance of Endpoint

The main computation and communication overheads of the endpoint are on preprocessing and token encryption.

1) Performance of the Preprocessing Phase in the First Session: We consider the time and the communication costs of the preprocessing phase executed on the endpoint side.

Time Cost: Table IV illustrates the preprocessing time on the endpoint side,[7] which shows that PrivBox requires similar time as other systems on the endpoint side. It is interesting to note that for a session with 3,000 rules, the PrivBox endpoints incur less than one second to prepare obfuscated rules. This is because the endpoints of our system only need to compute two exponentiation and a multiplication, which is quite lightweight so we believe it can also be applied to computation-constraint devices such as mobile and IoT devices.

Bandwidth Cost: In term of bandwidth, since there exists no discernible difference between the endpoint and **MB** (whatever sent by the endpoints will be received by **MB**), the bandwidth of the endpoint is the same as that of **MB** (cf. Fig. 9(b)).

2) Performance of the Token Encryption Phase: Table V shows the time cost for token encryption in the first session. For one token, PrivBox takes a little more time than other systems. For 800 tokens, PrivBox is approximately 1.3x$\sim$1.73x slower than other systems. This is mainly due to the following reason. To retain similar privacy guarantee as BlindBox (while other systems fail to retain such guarantee), PrivBox employs a more sophisticated mechanism (i.e., dual double-masking obfuscated rule generation) for the generation of obfuscated rules. As a result, the corresponding token encryption takes more time than other systems. As noted in [27], repeating tokens are likely to occur in actual communications, especially for the case where **R** having similar content each time it is accessed by **S**. Also, it is very likely that keywords in documents or articles tend to have a high frequency of common english words such as "encryption", "decryption". Fig. 11(a), (b) and (c) depict the improved performance of token encryption for a token set of 800 as the number of repeated tokens increases for three cases: (a) when the number of repeated token (repeating for four time) increases, as shown in Fig. 11(a), (b) when the number of times a token repeating itself increases, as shown in Fig. 11(b); and (c) when the percentage of (repeated) token being repeated from the previous session increases, as shown in Fig. 11(c). As shown in Fig. 11(a), (b) and (c), the more repeated tokens exist, the less the token encryption time of PrivBox consumes as compared to other systems. In particular, PrivBox outperforms PrivDPI when the percentage of one token repeating exceeds around 93% for a token set of 800.

[7]For Pine, since the preprocessing protocol is run between the gateway and the middlebox, we here record the running time of the gateway.

D. Performance of a Round Trip

We also perform the experiment for round trip time to evaluate the trade off between preprocessing and token encryption. We record the time that it takes to send a payload from **S** and further receive a message of the same length from **R**. The whole process of such one round trip mainly consists of the preprocessing phase, the token encryption phase, and the traffic inspection.

1) Performance in a Different Number of Rules: In this experiment, we set the number of tokens as 8,000, and from Fig. 12(a) we find that the performance gap between PrivBox and two prior works PrivDPI and Pine is very small, only about 1.3 seconds. When supporting multiple sessions, this gap becomes smaller, which is about 0.7 seconds from our results shown in Fig. 12(b). It is because our scheme supports reusability across encrypted tokens – some intermediate values for repeated tokens can be reused across different sessions, which further reduces the delay in sending a message. In real use, this small gap is hard to perceive so we believe PrivBox meets the scalability requirement in handling large rule sets.

2) Performance in a Different Number of Tokens: Here, the number of tokens can represent the size of data traffic in a session. If a session has a large traffic, e.g., sending a long message, more tokens would be produced. So this experiment can be also used to evaluate the latency under high-throughput traffic. Fig. 12(c) shows the time of one round trip when the number of (no repeating) token varies from 500 to 10,000 for a ruleset of 3,000 in the first session. For a token set of 10,000, PrivBox is approximately 79.5% slower than PrivDPI and approximately 75.2% slower than Pine. In contrast, P2DPI takes much more time than other systems, this is due to the fact that the traffic inspection of P2DPI incurs much more time than other three systems. Fig. 12(d) presents the one round trip time of a subsequent session for a ruleset of 3,000. For a token set of 10,000 in a subsequent session, PrivBox is approximately 65.6% slower than PrivDPI and approximately 59.3% slower than Pine. This shows that the round trip time of PrivBox in a subsequent session is closer to PrivDPI and Pine.

We also evaluated the scenario where token repeating exists for a token set of 8,000. We here consider the same cases as

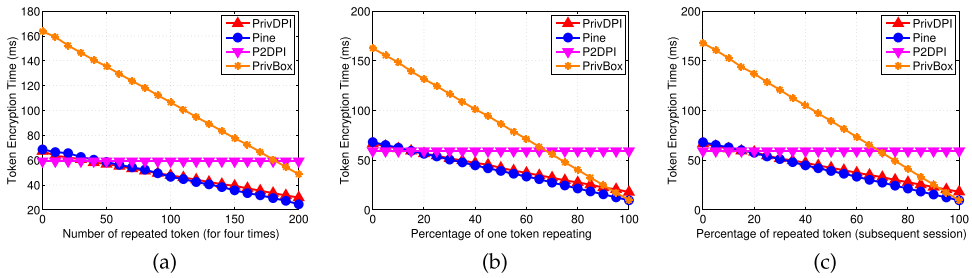


Fig. 11. Experimental results of Token Encryption for Repeated Token.

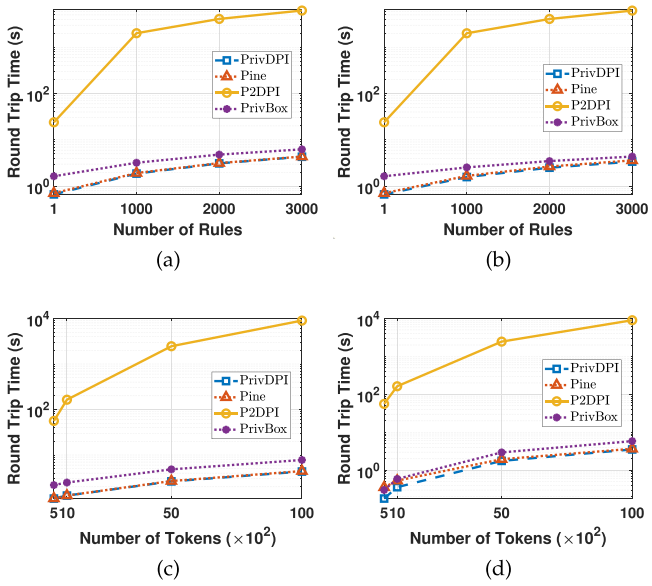


Fig. 12. Round trip time with a different number of rules and tokens (subfigures (a) and (c) are for the first session; subfigures (b) and (d) are for the subsequent session).

TABLE VI
ROUND TRIP TIME FOR A TOKEN SET OF 8,000

No. of Repeated Tokens (for 4 times)	Time (s)			
	PrivDPI	Pine	P2DPI	PrivBox
500	3.467	3.482	5,956.377	6.016
1,000	3.296	3.261	5,956.662	5.438
1,500	3.123	3.039	5,955.719	4.844
2,000	2.927	2.824	5,956.607	4.245

in the token encryption discussed above, and present the results in Tables VI, VII and VIII. As shown in Table VI, PrivBox runs approximately 45% slower than PrivDPI and 50.3% slower than Pine when the number of repeated token (repeating for four times) is 2,000 for a token set of 8,000. In Table VII, we may see that PrivBox is only approximately 38.1% slower than PrivDPI and approximately 45.3% slower than Pine when the percentage of one token repeating $\geq$85%. For a subsequent session, PrivBox is faster than PrivDPI when the percentage of repeated token (from previous sessions) $\geq$92% and faster than Pine when the percentage of repeated token (from previous

TABLE VII
ROUND TRIP TIME FOR A TOKEN SET OF 8,000

Percentage of One Tokens Repeating	Time (s)			
	PrivDPI	Pine	P2DPI	PrivBox
5	3.613	3.67	5,955.927	6.452
50	3.176	3.126	5,956.427	5.067
85	2.845	2.704	5,956.627	3.931
100	2.701	2.532	5,957.492	3.46

TABLE VIII
ROUND TRIP TIME FOR A TOKEN SET OF 8,000 (TOKEN REPEATED IN SUBSEQUENT SESSION)

Percentage of Repeated Tokens	Time (s)			
	PrivDPI	Pine	P2DPI	PrivBox
5	2.701	2.974	5,954.779	4.707
50	2.255	2.379	5,954.628	3.156
91	1.853	1.872	5,954.839	1.867
92	1.846	1.864	5,955.339	1.835
100	1.751	1.767	5,954.637	1.593

sessions) $\geq$91% (c.f. Table VIII). This indicates that for those scenarios where token repeating frequently happens, PrivBox is more practical than PrivDPI. For other scenarios where tokens are required changed (i.e., with token reuse $\leq$92%) such as intrusion prevention in the dynamic environment, one can be suggested to process token encryption in the pipeline. That is, for those tokens to repeat in the next session, the system can run token encryption in the current session instead of moving them to the next session. This will mitigate a long delay and lead to a faster round trip time for the next session. Alternatively, one can encrypt tokens with the aid of multi-threaded processors. By the Single Instruction Multiple Data (SIMD) technique, the system can generate encrypted tokens in parallel, which can also effectively speed up the token encryption protocols.

VIII. RELATED WORK

The most relevant works, which we have discussed in Section I, are BlindBox by Sherry et al. [31] and PrivDPI by Ning et al. [27]. These protocols enable privacy-preserving packet inspection directly over encrypted traffic by tokenizing and encrypting the payload. In particular, BlindBox is the first privacy-preserving DPI system that uses searchable encryption technique. The tokenization mechanism was later extended to include prefix matching in Embark by Lan et al. [22], which

caters for inspections under different services such as IP firewall, NAT, HTTP Proxy, data exfiltration and intrusion detection. Embark introduces a fully trusted gateway at the sender to perform the tokenization and encryption. However, unlike conventional (symmetric) searchable encryption, DPI asks for higher functionality and security. A trusted key-generation center produces the private key in searchable encryption and it is used both in encryption and token generation. Instead, in DPI, the private key can only be known to endpoints so an additional procedure is required to generate obfuscated rules without knowing this key, i.e., preprocessing protocol. This leads to an additional security requirement beyond the confidentiality of keywords and documents as in searchable encryption.

To mitigate the computation overhead of BlindBox, Canard et al. [8] introduced a token-matching protocol utilizing pairing-based public key operation, which is not compatible with current TLS protocol. SPABox [15] is another system that utilizes public key operation, which is mainly based on homomorphic encryption. A more efficient system was later proposed by Yuan et al. [43] by utilizing a high-performance encrypted filter, but it requires the server to register with the administration service of the enterprise hosting the client. A system called SplitBox [4] utilized two cloud servers to perform traffic inspection, in which every rule is XOR with a random string and then split into many blocks to the various middleboxes resided in one of the cloud servers. The performance of BlindBox was recently improved by PrivDPI [27], which introduces a reusable obfuscation mechanism that generates intermediate values to be reused across subsequent sessions. The efficiency of PrivDPI was further enhanced by Pine [26]. The property of rule hiding is also supported in Pine, which addresses the concern on the third-party cloud DPI setting. A more recent proposal P2DPI [21] showed that PrivDPI doesn't retain the same privacy of BlindBox as it can be compromised by the rule generator. As shown in Section III, P2DPI also cannot guarantee the same level of privacy as BlindBox. This work aims to provide a privacy-preserving DPI system that tackles the privacy concern while maintaining comparable performance as PrivDPI.

Another line of work on encrypted traffic inspection utilizes machine learning technique to analyze the meta data of the encrypted traffic without directly inspecting the encrypted payload. This includes the proposal by Anderson et al. [2], [3], [5] and Shi et al. [32]. Besides the technique of machine learning, trusted execution environment such as Intel SGX has also been deployed for deep packet inspection, including SGX-Box [19], SafeBricks [29], ShieldBox [35] and LightBox [13]. Compared to cryptographic schemes, the private and sensitive data can be processed within the enclave directly, so their performance can be drastically increased. As reported in LightBox [13], the packet I/O can achieve up to 10 Gbps. However, due to the need for the trusted hardware, these schemes introduces additional equipment costs and are difficult to be outsourced or virtualized. Moreover, TEE requires specialized code designs for different applications and may suffer from side-channel attacks [41]. Therefore, users are left with a tradeoff between efficiency and security, which depends on the real use. More recently, Grubbs et al. [18] initiated research on zero-knowledge middleboxes, which utilizes zero-knowledge proof to enforce network usage policies on encrypted traffic.

Independently, there are proposals that address the issue of accelerating inspection [42], [44] and authenticating middleboxes that perform inspection on the encrypted traffic, such as the accountable model proposed by Bhargavan et al. [6], mcTLS [25], mbTLS [24] and maTLS [23]. In these works, the middlebox is able to decrypt the encrypted traffic, but it should be authenticated by both endpoints to ensure it is a legitimate and approved middlebox. Such accountable model also proposes decryption and read/write authorizations based on the services to be performed by the middleboxes. The focus of our work is different from theirs in that we address the privacy of the payload during traffic inspection, while these proposals consider the authentication of middleboxes.

IX. Conclusion

We presented PrivBox, a DPI system that inspects encrypted traffic directly, which retains the same privacy guarantee as BlindBox and comparable efficiency to PrivDPI. The key technique behind PrivBox is the dual double-masking obfuscated rule generation that we proposed. The connection establishment time on the endpoint side of PrivBox is demonstrated to be similar as PrivDPI. The limitation of PrivBox is that the token encryption is around 1.46x slower than PrivDPI for non-repeating token. By reusing the session token(s) generated previously, the token encryption delay can be further reduced. PrivBox runs faster than PrivDPI for a set of 8,000 tokens with more than 92% already appeared in previous session(s). Overall, PrivBox is applicable to cases that require short and frequently established sessions, especially scenarios in which token repeating is common.

References

[1] National Security Agency, "Managing risk from transport layer security inspection," 2019. [Online] Available: https://media.defense.gov/2019/Dec/16/2002225460/-1/-1/0/INFO%20SHEET%20%20MANAGING%20RISK%20FROM%20TRANSPORT%20LAYER%20SECURITY%20INSPECTION.PDF

[2] B. Anderson and D. A. McGrew, "Identifying encrypted malware traffic with contextual flow data," in *Proc. 2016 ACM Workshop Artif. Intell. Secur.*, 2016, pp. 35–46.

[3] B. Anderson and D. A. McGrew, "Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity," in *Proc. ACM Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 1723–1732.

[4] H. J. Asghar, L. Melis, C. Soldani, E. D. Cristofaro, M. A. Kâafar, and L. Mathy, "SplitBox: Toward efficient private network function virtualization," in *Proc. ACM SIGCOMM Workshop Hot Top. Middleboxes Netw. Function Virtualization*, 2016, pp. 7–13.

[5] B. Anderson, S. Paul, and D. A. McGrew, "Deciphering malware's use of TLS (without decryption)," *J. Comput. Virology Hacking Techn.*, vol. 14, no. 3, pp. 195–211, 2018.

[6] K. Bhargavan, I. Boureanu, A. Delignat-Lavaud, P.-A. Fouque, and C. Onete, "A formal treatment of accountable proxying over TLS," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 339–356.

[7] H. Brenner, V. Goyal, S. Richelson, A. Rosen, and M. Vald, "Fast non-malleable commitments," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1048–1057.

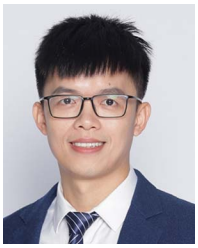
[8] S. Canard, A. Diop, N. Kheir, M. Paindavoine, and M. Sabt, "BlindIDS: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 561–574.

[9] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptol.*, vol. 13, no. 1, pp. 143–202, 2000.
[10] Cisco, "Cisco encrypted traffic analytics," 2021. [Online] Available: https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf
[11] X. de Carné de Carnavalet and M. Mannan, "Killed by proxy: Analyzing client-end TLS interception software," in *Proc. 23rd Annu. Netw. Distrib. Syst. Secur. Symp.*, 2016.
[12] M. Deng, K. Zhang, P. Wu, M. Wen, and J. Ning, "DCDPI: Dynamic and continuous deep packet inspection in secure outsourced middleboxes," *IEEE Trans. Cloud Comput.*, vol. 11, no. 4, pp. 3510–3524, Fourth Quarter 2023.
[13] H. Duan, C. Wang, X. Yuan, Y. Zhou, Q. Wang, and K. Ren, "LightBox: Full-stack protected stateful middlebox at lightning speed," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 2351–2367.
[14] Z. Durumeric et al., "The security impact of HTTPS interception," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017.
[15] J. Fan, C. Guan, K. Ren, Y. Cui, and C. Qiao, "SPABox: Safeguarding privacy during deep packet inspection at a MiddleBox," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3753–3766, Dec. 2017.
[16] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. Adv. Cryptol.*, 1986, pp. 186–194.
[17] V. Goyal, "Constant round non-malleable protocols using one way functions," in *Proc. ACM Symp. Theory Comput.*, 2011, pp. 695–704.
[18] P. Grubbs, A. Arun, Y. Zhang, J. Bonneau, and M. Walfish, "Zero-knowledge middleboxes," *IACR Cryptol. ePrint Arch.*, 2021. [Online] Available: https://eprint.iacr.org/2021/1022
[19] J. Han, S. M. Kim, J. Ha, and D. Han, "SGX-box: Enabling visibility on encrypted traffic using a secure middlebox module," in *Proc. 1st Asia-Pacific Workshop Netw.*, 2017, pp. 99–105.
[20] J. Jarmoc, "Transitive trust: SSL/TLS interception proxies," 2012. [Online] Available: https://www.secureworks.com/research/transitive-trust
[21] J. Kim, S. Camtepe, J. Baek, W. Susilo, J. Pieprzyk, and S. Nepal, "P2DPI: Practical and privacy-preserving deep packet inspection," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2021, pp. 135–146.
[22] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2016, pp. 255–273.
[23] H. Lee et al., "maTLS: How to make TLS middlebox-aware?," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019.
[24] D. Naylor, R. Li, C. Gkantsidis, T. Karagiannis, and P. Steenkiste, "And then there were more: Secure communication for more than two parties," in *Proc. Int. Conf. Emerg. Netw. Experiments Technol.*, 2017, pp. 88–100.
[25] D. Naylor et al., "Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 199–212.
[26] J. Ning et al., "Pine: Enabling privacy-preserving deep packet inspection on TLS with rule-hiding and fast connection establishment," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2020, pp. 3–22.
[27] J. Ning, G. S. Poh, J.-C. Loh, J. Chia, and E.-C. Chang, "PrivDPI: Privacy-preserving encrypted traffic inspection with reusable obfuscated rules," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1657–1670.
[28] McAfee Network Security Platform, 2020. [Online] Available: http://www.mcafee.com/us/products/network-security-platform.aspx
[29] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "SafeBricks: Shielding network functions in the cloud," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2018, pp. 201–216.
[30] C.-P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, 1991.
[31] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep packet inspection over encrypted traffic," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 213–226.
[32] R. Shi, H. Sun, L. Lan, Z. Peng, and C. Wang, "POSTER: A two-stage encrypted cryptomining traffic detection mechanism in campus network," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024.
[33] Emerging Threats: Open Source Signatures, 2020. [Online] Available: https://rules.emergingthreats.net/open/snort-2.9.0/rules/
[34] Snort, 2019. [Online] Available: https://www.snort.org/downloads/
[35] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "ShieldBox: Secure middleboxes using shielded execution," in *Proc. ACM Symp. SDN Res.*, 2018, pp. 2:1–2:14.
[36] US-CERT, "HTTPS interception weakens TLS security (alert TA17–075A)," 2017. [Online] Available: https://www.us-cert.gov/ncas/alerts/TA17-075A
[37] L. Waked, M. Mannan, and A. M. Youssef, "The sorry state of TLS security in enterprise interception appliances," 2018, *arXiv: 1809.08729*. [Online] Available: https://arxiv.org/abs/1809.08729
[38] P. Wu, R. Deng, Q. Shen, X. Liu, Q. Li, and Z. Wu, "ObliComm: Towards building an efficient oblivious communication system," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2331–2348, Sep./Oct. 2021.
[39] P. Wu, Q. Li, J. Ning, X. Huang, and W. Wu, "Differentially oblivious data analysis with Intel SGX: Design, optimization, and evaluation," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 6, pp. 3741–3758, Nov./Dec. 2022.
[40] P. Wu, J. Ning, X. Huang, and J. K. Liu, "Differentially oblivious two-party pattern matching with sublinear round complexity," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 5, pp. 4101–4117, Sep./Oct. 2023.
[41] P. Wu, J. Ning, J. Shen, H. Wang, and E.-C. Chang, "Hybrid trust multi-party computation with trusted execution environment," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022.
[42] H. Xu, H. Chang, K. Qiu, Y. Hong, W. Zhu, and X. Wang, "Accelerating deep packet inspection with SIMD-based multi-literal matching engine," *IEEE Trans. Netw. Service Manag.*, to be published, doi: 10.1109/TNSM.2024.3354985.
[43] X. Yuan, X. Wang, J. Lin, and C. Wang, "Privacy-preserving deep packet inspection in outsourced middleboxes," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[44] C. Zhou et al., "NetDPI: Efficient deep packet inspection via filtering-plus-verification in programmable 5G data plane for multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 15031–15047, Dec. 2024.

Pengfei Wu (Member, IEEE) received the PhD degree in software engineering from Peking University, Beijing, China, in 2020. Now, he is working as a research scientist with the School of Computing and Information Sciences, Singapore Management University. Before joining SMU in September 2023, he was a research fellow with the School of Computing, National University of Singapore. His research interests include cloud security and Big Data security.

Jianting Ning (Member, IEEE) received the PhD degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, in 2016. He is currently with the School of Cyber Science and Engineering, Wuhan University, Wuhan, China, Faculty of Data Science, City University of Macau, Macau, China, and also with the College of Computer and Cyber Security, Fujian Normal University, Fuzhou, China. He has published papers in major conferences or journals, such as ACM CCS, NDSS, and the *IEEE Transactions on Dependable and Secure Computing*. His research interests include applied cryptography and information security.

Xinyi Huang received the PhD degree from the School of Computer Science and Software Engineering, University of Wollongong, Australia, in 2009. He is currently an associate professor with the Thrust of Artificial Intelligence, Information Hub, Hong Kong University of Science and Technology (Guangzhou), China. His research interests include cryptography and information security. He has published more than 160 research papers in refereed international conferences and journals, such as CCS, Crypto, and the *IEEE Transactions on Information Forensics and Security*.



Rongmao Chen received the PhD degree in computer science from the University of Wollongong, Australia. He is now a full professor with the College of Computer Science and Technology, National University of Defense Technology in China. His major research interests include applied cryptography and cybersecurity. He has published more than 70 research papers in refereed international conferences and journals, such as Crypto, the *IEEE Transactions on Information Forensics and Security* and *IEEE Transactions on Dependable and Secure Computing*.

Kai Zhang received the bachelor's degree in computer science and technology from Shandong Normal University, China, in 2012, and the PhD degree in computer science and technology from East China Normal University, China, in 2017. He visits Nanyang Technological University in 2017. He is currently a professor with the Shanghai University of Electric Power, China. His research interests include applied cryptography and information security.

Kaitai Liang (Member, IEEE) received the PhD degree in computer science from the Department of Computer Science, City University of Hong Kong. He worked as a post-doctoral researcher with Aalto University and held a permanent assistant professor position in secure systems with the University of Surrey. He has published a series of research works, applying information security and crypto tools to address cybersecurity challenges. These publications have appeared in high-tier international information security journals and conferences, e.g., USENIX Security, NDSS, the *IEEE Transactions on Dependable and Secure Computing*.