P2DPI: Practical and Privacy-Preserving Deep Packet Inspection

Jongkil Kim[1], Seyit Camtepe[2], Joonsang Baek[1], Willy Susilo[1], Josef Pieprzyk[2,3], and Surya Nepal[2]

[1] University of Wollongong, Australia
[2] CSIRO Data61, Australia
[3] Institute of Computer Science, Polish Academy of Sciences, Poland
Jongkil@uow.edu.au, Seyit.Camtepe@data61.csiro.au, Baek@uow.edu.au, Wsusilo@uow.edu.au, Josef.Pieprzyk@data61.csiro.au, Surya.Nepal@data61.csiro.au

Abstract. The amount of encrypted Internet traffic almost doubles every year thanks to the wide adoption of end-to-end traffic encryption solutions such as IPSec, TLS and SSH. Despite all the benefits of user privacy the end-to-end encryption provides, the encrypted internet traffic blinds intrusion detection system (IDS) and makes detecting malicious traffic hugely difficult. The resulting conflict between the user's privacy and security has demanded solutions for deep packet inspection (DPI) over encrypted traffic. The approach of those solutions proposed to date is still restricted in that they require intensive computations during connection setup or detection. For example, BlindBox, introduced by Sherry et al. (SIGCOMM 2015) enables inspection over the TLS-encrypted traffic without compromising users' privacy, but its usage is limited due to a significant delay on establishing an inspected channel. PrivDPI, proposed more recently by Ning et al. (ACM CCS 2019), improves the overall efficiency of BlindBox and makes the inspection scenario more viable. Despite the improvement, we show in this paper that the user privacy of Ning et al.'s PrivDPI can be compromised entirely by the rule generator without involving any other parties, including the middlebox. Having observed the difficulties of realizing efficiency and security in the previous work, we propose a new DPI system for encrypted traffic, named "Practical and Privacy-Preserving Deep Packet Inspection (P2DPI)". P2DPI enjoys the same level of security and privacy that BlindBox provides. At the same time, P2DPI offers fast setup and encryption and outperforms PrivDPI. Our results are supported by formal security analysis. We implemented our P2DPI and comparable PrivDPI and performed extensive experimentation for performance analysis and comparison.

Keywords: Deep Packet Inspection; Searchable Encryption; Intrusion Detection System; Exfiltration System

1 Introduction

Network middlebox systems are widely adopted to protect both network operators and end users. Network intrusion detection, intrusion prevention and exfiltration systems are typical examples of middlebox solutions protecting networks from internal and external attacks. Middlebox systems execute deep packet inspection (DPI) to evaluate packet payloads and examine their compliance to policies (e.g., a security policy and an information management policy). Once a non-compliance is detected, a middlebox triggers an appropriate action to maintain the network's security. In particular, intrusion detection systems (IDS) and intrusion prevention systems (IPS) identify and prevent malicious attempts such as malware intrusions and security policy violations by monitoring network traffic. They raise an alert for security auditors and perform automated protective actions following a predefined set of rules. Exfiltration systems identify specific signatures such as document watermarks and prevent the leakage of the company's intellectual properties.

Although DPI takes a vital role as a primary security control over the organizational network, the demand for user privacy significantly restricts the usage of DPI. Nowadays, almost all traffic on the Internet is encrypted. According to Google trend [1], all top 100 sites on the Internet support the HTTPS, which offers end-to-end encryption. Among them, 96 websites use the HTTPS as default. Moreover, this is an on-going trend as the concerns on user privacy increase. DPI's function is significantly disrupted by the end-to-end encryption, such as the TLS protocol, because encryption makes the traditional packet inspection infeasible. When the end-to-end encryption is applied to the traffic, only two ends (e.g., a client and a server) can share the contents. Any system in the middle is inaccessible to what is encrypted.

Due to this, the middlebox technologies (see [21, 18, 10, 22]) naturally evolve to inspect encrypted traffic. A well-known solution in this direction is using "interception proxy" [10, 17, 9], which behaves like an adversary in the man-in-the-middle (MITM) attack. The solution interrupts the authentication process between a client and a server. Therefore, a proxy breaks an end-to-end secure connection into two parts: user-to-proxy (i.e. the proxy behaves as the server to the client) and proxy-to-server (i.e. the proxy plays the role of the client to the server). It may not work well if either the client or the server uses digital certificates [10] with a well-established certificate chain. Moreover, this approach compromises user privacy since all the traffic between a client and a server is accessible at the proxy, which is a third party. These downsides became a significant concern in taking this approach.

Dixon et al. [6] proposed an alternative deployment model where all middlebox functionalities are performed at the host by the end user to control and monitor their traffic. This host-based model does not hide the rules from the end user. However, hiding the rules is considered as an important compatibility requirement for a modern IDS. First of all, according to Paxson [20], hiding the detection rules from the end user makes evading IDS at the user more difficult for an attacker. Second, according to [22], the detection rule sets are important

intellectual property for IDS vendors. Maintaining their rule sets more competitive in the market can be one of the essential business priorities of the vendors. Therefore, keeping the detection rules secret is important from the business point of view.

Because of the apparent disadvantages of an intercepting proxy, researchers seek another approach that allows deep packet inspection while preserving user privacy. They observed that these two controversial properties, DPI and protecting user's privacy, can be achieved if encryption permits searching – this is the well-known searchable encryption [24, 11, 7]. Searchable encryption enables middlebox systems for pattern matching of encrypted packets with encrypted search keywords or tokens [13, 8, 22, 5]. Note that in this solution, all search operations are done without decryption of encrypted packets. Moreover, search tokens are generated only by negotiating them with users. Therefore, only limited numbers of patterns of violation approved by the users are matched with the encrypted traffic for the detection. This constraint enforces the middleboxes do the task only allowed to them by users and minimizes the potential leakage of user's private data.

Unfortunately, such middlebox solutions are still immature and suffer from their drawbacks. The main drawback is low performance due to a heavy computational overhead. The existing solutions require a slow set up time (around 97s) for an initial connection [22, 13] or impractical inspection speed performance [5].

Recently, PrivDPI is proposed by Ning et al. [18]. It significantly improves the efficiency of Sherry et al's BlindBox [22] using reusable obfuscation. However, unlike their claim, it does not achieve the same privacy level as the contents communicated between client and server can be compromised by a third party. We demonstrate in a later section that PrivDPI does not guarantee the same level of privacy as BlindBox does by presenting an attack that compromises the security of "Preprocessing Protocol". Our attack is based on the fact that a rule generator in PrivDPI plays more than merely creating rules to execute a simple exhaustive message search attack on the encrypted traffic, which did not apply to Sherry et al.'s BlindBox.

Therefore, constructing a deep packet inspection system that is efficient enough for practical use, but preserves user privacy by providing end-to-end encryption which does not allow abuse of inspection capability leaves as an essential problem.

1.1 Our Contributions

We provide a new protocol, which we call "Practical and Privacy-preserving Deep Packet Inspection (P2DPI)". P2DPI aims to offer the same level of privacy that BlindBox [22] does. Therefore, it guarantees that any third party, including a rule generator and a middlebox, cannot compromise the privacy of two end users (e.g., a client and a server). Moreover, the middlebox can only inspect a limited amount of information that is negotiated with the end users. At the same time, P2DPI is practical from an efficiency perspective as it does not have a significant delay when the connection is set-up.

Earlier, PrivDPI aimed to achieve those goals, retaining security and privacy at the same level as those of BlindBox, but improving efficiency. However, we observed that PrivDPI is susceptible to an exhaustive message search attack, which did not apply to BlindBox. Therefore, P2DPI is the first protocol that *truly* achieves those goals at the same time. P2DPI uses a key-homomorphic pseudorandom function (KH-PRF) to exchange the obfuscated detection rules between a user and a middlebox without revealing their obfuscation keys. Employing KH-PRF brings a significant improvement to the DPI system. In particular, P2DPI turns out to be more efficient than BlindBox while maintaining the same security level. Compared to PrivDPI, P2DPI significantly improves both security and efficiency.

In particular, our paper presents followings:

- **Security Analysis on PrivDPI [18]**: We provide our security analysis on PrivDPI. We show that PrivDPI is susceptible to an exhaustive message search attack. That is, in PrivDPI, if a binary message (0 or 1) is encrypted, the adversary can easily tell which message was encrypted. This vulnerability enables a third party, who is the other than two end users, to permit exhaustive search over the messages. We will present a detailed analysis of the PrivDPI protocol in Section 3.
- Presenting **P2DPI**: We propose P2DPI, which resolves the security issues of PrivDPI. P2DPI retains the MBSE (middlebox searchable encryption) security for user privacy and provides formal security proof. Therefore, the user's private data cannot be compromised by any parties, including a middlebox and a rule generator. Thus, P2DPI achieves the same level of security as BlindBox does. Moreover, P2DPI is more efficient than PrivDPI. Compared to PrivDPI, the encryption time is 3.5 times faster. For example, P2DPI reduces encryption time from 55 ms to 16 ms for 1000 message tokens. The setup time of P2DPI is also slightly reduced due to the reduced computational overhead. The detection time of P2DPI keeps equivalent to that of the previous best solutions, PrivDPI and BlindBox.
- **Empirical analysis**: We provide empirical analysis on P2DPI. For the comparison, we implement both P2DPI and PrivDPI and compare the computational overhead. We used C language to implement the protocols, and python is partially used to simulate digital signatures.

We highlight that P2DPI does not have any downsides from security and efficiency perspectives when we compare it with PrivDPI [18].

This paper is organized as follows. An overview of our P2DPI together with a description of the relevant threat model is given in Section 2. In Section 3, we present an attack that compromises the user privacy of PrivDPI. In Section 4, we briefly describe concepts and cryptographic tools, which will be used in our protocol. Section 5 is the central part of the paper, which discusses components and construction of the proposed P2DPI protocol. Security and performance of the proposed protocol are analyzed in Sections 6 and 7, respectively. Section 8 provides an overview of related work. We conclude our work in Section 9.

2 System Overview

2.1 System Architecture

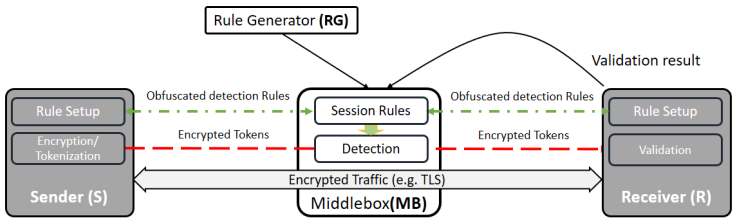

Fig. 1. P2DPI System.

P2DPI system consists of four parties: Rule generator (RG), Middlebox (MB), Sender (S) and Receiver (R) as depicted in Figure 1. The detailed descriptions of each party are as follows.

RG is a party that supplies detection rules to MB. RG communicates with MB only for deploying and updating the detection rules. Usually, RG is considered to be honest-but-curious. It generates the detection rules reliably, but it can behave maliciously or coerced by a malicious party to compromise users' privacy.

MB is an intrusion detection system or an exfiltration system, which inspects encrypted traffic sent from S to R. Organizations deploy MB in their networks. MB receives (encrypted) detection rules from RG. MB is honest-but-curious for the communication between S and R.

S and R are end users who communicate securely with each other using TLS-like protocols. They do not want to reveal the full content of their communication to MB nor RG. Our solution assumes that at least one of them, either S or R, is honest. Both S and R are curious about the detection rules. The roles of S and R are exchangeable since they usually establish a bidirectional channel. We assume that both S and R cannot be malicious at the same time. (That is, either S or R is honest at least.) This is a common assumption that IDS and exfiltration systems premise. We will discuss this in more details in the following subsection (Section 2.2).

As illustrated in Figure 1, P2DPI protocol between a sender (S) and a receiver (R) is initiated by exchanging detection rules in obfuscated format with MB. After MB and S/R successfully share the detection rules, MB computes session rules, which is used for the inspection only for this connection, from the obfuscated rules. S tokenizes traffic and encrypts each token, which is denoted as encrypted tokens to send them to R using the same encryption that is used in the session rules. Therefore, MB can match the detection rules to the packet payloads while both are encrypted. The encrypted tokens must be validated

whether they are truly matched with the data which are delivered via an encrypted channel (e.g., TLS encryption for HTTPS service). This can be verified that one of S and R, which is honest to MB. Note that we assume that R is honest in Figure 1.

The communications among MB, S and R in the P2DPI protocol are taking place together with TLS (Transport Layer Security), which is a transport layer protocol. However, inspection is not necessarily performed in the same layer. DPI is usually performed in the network layer rather than the transport layer as it needs direct access to the data (payload) and the header information of each packet [22]. From the inspection perspective, P2DPI is no different from the original DPI except that packet inspection is conducted over encrypted traffic using detection rules in an obfuscated format.

2.2 Threat Model

Typical Threats to MB (IDS): There are common threats to all IDS solutions [13, 5, 22, 18]. We cannot enforce security in any IDS system if both endpoints are malicious, i.e., both S and R collude for malicious activities. This is because the colluding endpoints may bypass any control using side channels [20] or place a backdoor in the communication system to encrypt traffic to hide its patterns. Note that assuming at least one endpoint is honest is the default in the previous work such as BlindBox [22] and PrivDPI [18].

We remark that the said assumption is reflected in the operation of current IDS systems. As exemplified in [22], many recent exfiltration detection systems exercise the prevention of "accidental exfiltration" to get rid of the situation when their users accidentally transmit a wrong file, such as a company's confidential document, as an email attachment. (If this case is removed, the users can be considered honest.) The assumption and exercise are based in the fact that users comply with the company's policy and do not have any intention to bypass or trick IDS systems, while incidents usually happen due to their careless behaviors as pointed out in Verizon's report [25]. From now on, we assume that the receiver R is the honest end-user.

The authors of BlindBox [22] point out that the detection rules in the IDS and exfiltration systems must be kept hidden from both S and R, which is also applied to our P2DPI system. Otherwise, S and R can manipulate the content to evade the detection rules used by the IDS and exfiltration systems. This is necessary even if we assume that one of S and R is honest. For example, a malicious website that lures inside workers to access their site may want to know detection rules to bypass the IDS.

Threats to Users (S and R): By collaborating RG, MB provides intrusion detection and exfiltration normally by inspecting the traffic (see [22]). We assume that MB and RG are curious about the contents of data sent from S and R. The users S and R want to preserve their communication confidentiality while agreeing for deep packet inspection. Here, S and R limit the inspection capability of MB and RG by restricting the number of detection rules. Therefore, MB and

RG execute the deep packet inspection without compromising the confidentiality of communication between S and R more than S and R agree.

2.3 Overview of Our Techniques

The key challenge in P2DPI is securely hiding the information between parties. MB and RG want to hide the detection rules from S and R. At the same time, S and R do not want to reveal their shared key to MB and RG. These requirements mandate that the detection rules must be delivered to S and R in an encrypted format[4], but S and R need to encrypt the detection rules using their key without knowing their content. In other words, S and R need to update the key for the encrypted detection rules given by MB and RG without decrypting them. Note that this is a well-known practice in cryptography, called "key-homomorphic encryption."

More precisely, let $C = \mathrm{Enc}(K_1, M)$ be a ciphertext, where a message M is encrypted using a secret key K_1. In key-homomorphic encryption, an updater updates C to $C' = \mathrm{Enc}(K_1 \cdot K_2, M)$ using a key K_2, which is owned by the updater, without decrypting C. Because the key-homomorphic encryption does not require decryption in this process and does not have the key K_1, the updater cannot get M. We will define key-homomorphic encryption more formally in Section 4.1.

We will use the key-homomorphic encryption to exchange the information without revealing each other's secret. Let K_{SR} be a shared key between S and R, and let M_{rule} be a detection rule. The biggest challenge in our protocol is that MB needs to know $\mathrm{Enc}(K_{SR}, M_{rule})$ without having K_{SR} or revealing M_{rule} to S and R. To achieve this, MB first computes $C_{dpi} = \mathrm{Enc}(K_{MB}, M_{rule})$ and asks S and R to update it to $C'_{dpi} = \mathrm{Enc}(K_{MB} \cdot K_{SR}, M_{rule})$. When MB receives C'_{dpi} back from S and R, it can compute $\mathrm{Enc}(K_{SR}, M_{rule})$ by computing $\mathrm{Enc}((K_{MB} \cdot K_{SR}) \cdot {K_{MB}}^{-1}, M_{rule})$ using its knowledge of K_{MB} and key homomorphism.

Note that key-homomorphic encryption algorithms already exist [3]. Therefore, we can simplify the DPI technique by adopting one of the existing ones. Moreover, those algorithms are proven to be secure so that they mathematically guarantee that the requirements we aforementioned are satisfied. In Section 4, we review a slightly generalized version, "key-homomorphic PRF" (which, of course, covers the definition key-homomorphic encryption).

3 Security Analysis of PrivDPI

In this section, we show that PrivDPI, recently proposed by Ning et al.[18], cannot achieve user privacy, contrary to their claim that "PrivDPI retains the same security and privacy guarantee" as BlindBox. (Section 1.1 of [18].)

[4] We slightly abused the term "encrypt" as the rules actually obfuscated rather than encrypted. However, we use this term in this subsection only as we explain obfuscation is achieved by Key-homomorphic encryption in a high-level view.

As described in Section 2.1, there are four parties involved in the typical setting of DPI protocols (including BlindBox, PrivDPI and ours): Sender (S), Receiver (R), Middlebox (MB) and Rule generator (RG). S and R are end users, who communicate with each other over an encrypted channel. MB inspects the channel using the detection rules generated by RG. In particular, S and R will send and receive a data token t by encrypting it using the same key k. MB will determine whether the encrypted traffic contains t by checking if the encrypted token of t is equivalent to the encrypted detection rule of r.

According to Sherry et al., their BlindBox protocol should not be used in a setting where *both* RG and MB are controlled (coerced) by an adversary [22]. However, they did mention that even if *one of* RG and MB is coerced by the adversary, end users in BlindBox should remain protected from the attack that their traffic is inspected by more rules than they have agreed. Informally, it can be achieved as follows.

Suppose that 1000 rules are agreed among RG, MB and S/R. Let $\mathcal{R}$ be a set that contains those obfuscated rules and $sig(\mathcal{R})$ be their signatures signed by RG. MB should be able to inspect the encrypted traffic using each rule $r \in \mathcal{R}$ only. (Note that S and R do not know the content of r.) This is possible in BlindBox through the following steps: 1) After verifying the signature associated with r, S provides MB with obfuscation of AES with the hard-coded key k; 2) MB can compute $\mathsf{AES}_k(r)$ without knowing k.

To have a feeling for the security of this scheme, consider the two cases below.

First, assume that MB is compromised, but RG is not. Since S and R can verify whether the detection rules $\mathcal{R}$ are from RG using their signatures $sig(\mathcal{R})$ and the public key of RG. MB cannot add any malicious detection rule on its own.

Second, assume that RG is compromised, but MB is not: It is also impossible for RG to compute $\mathsf{AES}_k(r')$ for a new rule $r' \notin \mathcal{R}$, since RG does not know k. RG cannot generate it on its own. Hence, as long as MB refuses to include r' in $\mathcal{R}$ (secretly), the privacy for S and R is attained.

Now, we investigate the security of Ning et al.'s PrivDPI [18]. For a clear exposition, we describe the "Preprocessing" protocol of PrivDPI, which is used to setup obfuscated rules among S, R and MB, as follows. (Note that we only changed the notions for users, C (Client) and S (Server) in [18] to S and R, respectively. Other notations remain as they are.)

Preprocessing Protocol of PrivDPI [18]

Input: MB has inputs $(\{s_i, R_i, sig(R_i)\}_{i\in[N]})$ (from RG), where $R_i = g^{\alpha r_i + s_i}$ (here, r_i is a rule, and α and s_i are values for "blinding" r_i); S and R have common inputs k.

1. S computes $K_S = g^k$ (using her k), and sends K_R to MB. Similarly, R computes $K_R = g^k$ (using her k) and sends K_R to MB.
2. MB checks whether K_S equals K_R . If not, it halts and outputs $\perp$. Otherwise, it sends $(\{R_i, Sig(R_i)\}_{i\in[N]})$ to S.
3. S does as follows:

(a) For $i \in [N]$, check if $Sig(R_i)$ is a valid signature on R_i using the public key of the signature scheme used by RG. If not, halt and output $\perp$.
(b) Compute $K_i = (R_i \cdot K_S)^k = g^{k\alpha r_i + k s_i + k^2}$ for $i \in [N]$. Finally, send $\{K_i\}_{i\in[N]}$ to MB.

4. For $i \in [N]$, MB verifies whether the equation $e(K_i, g) = e(R_i \cdot K_S, K_S)$ holds. If not, it halts and outputs $\perp$. Otherwise, for $i \in [N]$, it calculates the reusable obfuscated rule $I_i = K_i/(K_S)^{s_i} = g^{k\alpha r_i + k^2}$ for future use.

RG's attack proceeds as follows. RG simply observes the communication between MB and S in the above protocol, then it obtains K_S and $K_i = g^{k\alpha r_i + k s_i + k^2}$ from it. This is possible because those parameters are not encrypted. However, since RG knows α, r_i and s_i, it can get g^{k^2} by computing $K_i/K_S^{\alpha r_i + s_i}$. The consequence of getting this value is that RG can compute the additional session rule S'_i for any rule r'_i by computing $S'_i = K_S^{\alpha r'_i} g^{k^2} = g^{k\alpha r'_i + k^2}$. In other words, RG can produce a session rule for a new rule which is different from those processed in the Preprocessing Protocol. Therefore, PrivDPI does not guarantee the same security that BlindBox provides.

Moreover, this attack can be extended to compromise the middlebox searchable encryption (MBSE) security through an "exhaustive message search" attack: RG selects a rule representing low entropy data such as temperature, date or even binary number and creates an obfuscated rule on its own. Then it exhaustively searches the encrypted traffic for the corresponding message (i.e., its decryption) that matches the rule. The consequence of this attack is that RG can break the indistinguishability of two sequences of the encrypted traffic, which compromises the MBSE security. (Note that MBSE security is formally defined in Section 6.)

The difference in processing rules between BlindBox and PrivDPI is that in BlindBox, RG creates rules only, while in PrivDPI, RG generates extra values along with the rules used in the preprocessing protocol which involve MB and end users. Hence, as long as MB refuses to process RG's rules that were not agreed, the user privacy is achieved in BlindBox, However, RG in PrivDPI can compromise the rule processing *on its own* even if MB refuses to collude with RG.

The above attack implies that there is a problem in the analysis of the MBSE security of PrivDPI. Note that MBSE security does not have any restriction as to who can be an adversary, that is, it guarantees the user privacy from *any* adversary, including MB and RG. In the attack game for MBSE security, the adversary generates rules and queries them (to the challenger) to get obfuscated rules. This reflects the real situation of BlindBox, where RG does not generate any other values except rules. Hence, the security proof of BlindBox [23] works for the case in which RG or MB can be an adversary. However, in the preprocessing protocol of PrivDPI, RG generates its internal values like α and s_i, which are not given to the adversary in the MBSE security proof for PrivDPI [18]. In other words, their proof does not address the real privacy issues that can arise in PrivDPI.

One may argue that RG must be fully trusted. However, this is not the case, as it is assumed to be some external party like a non-profit generation, and it can be coerced by an adversary (such as a totalitarian government), according to Sherry et al. [22]. Also, we envision that in terms of functionality for DPI in general, making RG generate parameters that will be used by other parties, including MB, in a fully-trusted way is cumbersome. Contrary to the authors' claim, PrivDPI did not achieve the same level of the security that BlindBox delivers because BlindBox does not allow this attack and does not give trust to RG while PrivDPI does.

4 Preliminaries

In this section, we overview main techniques for designing our protocol, review cryptographic primitives and summarize various notations, which will be used to describe our protocol.

4.1 Key-homomorphic PRF

Naor et al.'s pseudorandom function (PRF) based on the DDH (Decisional Diffie-Hellman) assumption [16], which we call "key-homomorphic PRF", is an essential component of the P2DPI protocol. First, we review the definition of PRF as follows.

Definition 1 (Pseudorandom Function (PRF) [3]) *An efficiently computable function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, where $\mathcal{K}$ is a key space, $\mathcal{X}$ is an input space and $\mathcal{Y}$ is an output space, is PRF if no efficient adversary can distinguish F from a truly random function given only a black-box access to F.*

We also review the definition of the DDH assumption as follows.

Definition 2 (*DDH* assumption) *Let $\mathbb{G}_p$ be a finite cyclic group of prime order p, generated by g. The DDH assumption holds if, given $\{g, g^{c_1}, g^{c_2}, T \in \mathbb{G}_p\}$ for uniform c_1 and c_2 from $\mathbb{Z}_p$, there is no probabilistic polynomial time (PPT) algorithm that can determine whether T is $g^{c_1 c_2}$ or a random integer from $\mathbb{G}_p$ with non-negligible probability.*

Now, we review the definition of key-homomorphic PRF.

Definition 3 (Key-Homomorphic Function [16]) *Suppose that $k \in \mathbb{Z}_p$ is chosen uniformly at random. Also, specify a one-way function $H : \mathcal{X} \to \mathbb{G}_p$. Then, the key-homomorphic function F_{KH} is defined as follows:*

$$F_{\mathrm{KH}}(k, x) := H(x)^k.$$

The above key-homomorphic function has the following properties, which are all proven in [3].

Property 1. (PRF): If H is modelled as a random oracle, the key-homomorphic function F_{KH} is PRF. That is, the advantage of a PPT adversary $\mathcal{A}$ in distinguishing F_{PRF} from a random function is negligibly small.

Due to the above property, we call the key-homomorphic function as "key-homomorphic PRF".

Property 2. (Key Homomorphism): For any $k_1, k_2 \in \mathbb{Z}_p$ and any $x \in \mathcal{X}$,

$$F_{\mathrm{KH}}(k_1, x) \cdot F_{\mathrm{KH}}(k_2, x) = F_{\mathrm{KH}}(k_1 + k_2, x).$$

Property 3. (Commutativity): For any $k_1, k_2 \in \mathbb{Z}_p$ and any $x \in \mathcal{X}$,

$$\begin{aligned} F_{\mathrm{KH}}(k_1, x)^{k_2} &= \underbrace{F_{\mathrm{KH}}(k_1, x) \cdot \ldots \cdot F_{\mathrm{KH}}(k_1, x)}_{k_2 \text{ times}} \\ &= F_{\mathrm{KH}}(\underbrace{k_1 + \ldots + k_1}_{k_2 \text{ times}}, x) \\ &= F_{\mathrm{KH}}(k_1 \cdot k_2, x). \end{aligned}$$

Similarly, $F_{\mathrm{KH}}(k_2, x)^{k_1} = F_{\mathrm{KH}}(k_1 \cdot k_2, x)$.

Throughout the paper, we utilize $H(x, k) = (g^{H_1(x)}h)^k$ where $g, h \in \mathbb{G}_p$, $x \in \mathcal{X}$, $k \in \mathbb{Z}_p$ and $H_1(\cdot)$ is a programmable random oracle.[5] Note that some other constructions of key-homomorphic PRFs can be found in [16] and [3].

4.2 Notations

The P2DPI protocol uses various keys, each of which has a different purpose. We summarize them as follows.

- k_{MB}: This key is created by RG and securely shared with MB. It is used to hide the detection rules from S and R (by making them indistinguishable from random values of the same length). Note that disclosing this key will allow S and R to bypass the inspection by leaking detection rules.
- k_{SR}: This key is a shared key between S and R and used to obfuscate their communication. The disclosure of this key allows an adversary to potentially read packets sent from S to R through an exhaustive message search attack. This key is also used to update the key for obfuscating detection rules sent by MB. Its value is not revealed to MB during the update because of the key-homomorphic property of our protocol.
- k_{SSL}: This is a session key used for a secure channel established by the SSL/TLS protocol. We assume that this key is managed by the SSL/TLS protocol.

The detection rules in P2DPI can be represented using the following formats:

[5] in this version of the paper, the definition $H(x, k) = (g^{H_1(x)})^k$ in the previous version [12] is revised to $H(x, k) = (g^{H_1(x)}h)^k$ due to the malleability of the previous choice.

- r_i: A **detection rule** (in a plain text format), generated by RG.
- R_i: An **obfuscated detection rule** by RG. r_i is obfuscated to R_i using the key-homomorphic PRF by the key, k_{MB}, which is allocated to each MB.
- sig(R_i): A **signature of obfuscated detection rule** R_i using the private key of RG. The signature can be verified using RG's public key.
- I_i: An **intermediate obfuscated detection rule**. Because R_i is obfuscated using the key-homomorphic PRF, its key can be updated from k_{MB} to $k_{MB} \cdot k_{SR}$, here k_{SR} is a shared obfuscation key between S and R, which is agreed during the handshake process. I_i corresponds to r_i, but it is obfuscated using the updated key $k_{MB} \cdot k_{SR}$.
- S_i: A **session rule**, which is used during packet inspection. S_i corresponds to r_i, but it is obfuscated using the key k_{SR}, the shared key between S and R.

5 P2DPI Protocol

In this section, we describe our P2DPI protocol.

5.1 Rule Setup

In the rule setup stage, RG generates a secret key k_{MB} for MB, and S and R generate and share the other key k_{SR} to prepare a deep packet inspection. Also, k_{SSL} is created and shared for the SSL/TLS encryption channel between S and R.

The main idea of the rule setup algorithm (Algorithm 1) is to allow each party to obfuscate detection rules using their own keys so that the *rule hiding security* holds. This means that RG and MB obfuscate rules using k_{MB}, and users S and R additionally update the key of already obfuscated rules R_i's from k_{MB} to $k_{MB} \cdot k_{SR}$ using the key-homomorphic property. Now, knowing the intermediate obfuscated rules I_i's and the key k_{MB}, MB removes k_{MB} from I_i's and generates session rules S_i's, which are obfuscated under the key k_{SR} only. MB uses the session rules to inspect packets sent from S to R. Algorithm 1 describes the procedure.

5.2 Encryption

P2DPI requires the traffic $\mathcal{T}$ from S to R to be parsed to t_i and obfuscates to hide the content and match it against the session rules S_i for all $i \in [N_r]$. (Note that S_i's are negotiated and obfuscated in the rule setup (Algorithm 1).) Hence, the encryption phase involves two processes: extracting tokens and obfuscate tokens as follows

Extracting tokens: Our detection algorithm uses tokenization methodology of BlindBox [22].

Algorithm 1: Rule Setup

1. RG generates $g, h \in \mathbb{G}_p$ where $\mathbb{G}_p$ is a finite cyclic group of prime order p. It also selects 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 $i \in [N_r]$ where N_r is the total number of detection rules, compute
$$R_i = (g^{H_1(r_i)}h)^{k_{MB}}.$$
RG then signs R_i to $sig(R_i)$ using its private key and sends a set of detection rules $\{(R_i, sig(R_i)|i \in [N_r]\}$ to MB.
2. S and R send a connection request to MB.
3. MB sends $\{R_i, sig(R_i)|i \in [N_r]\}$ to S and R.
4. Both S and R verify $sig(R_i)$ 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
$$\{I_i = {R_i}^{k_{SR}} = (g^{H_1(r_i)}h)^{k_{MB} \cdot k_{SR}} | i \in [N_r]\}.$$
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 $\{S_i = (I_i)^{1/k_{MB}} = (g^{H_1(r_i)}h)^{k_{SR}} | i \in [N_r]\}$.

- *Windows-based tokenization* method extracts tokens from a packet using a simple sliding window algorithm. In particular, it parses a bytestream by a fixed length. We follow BlindBox and use 8-byte windows. For example, the words "strictly confidential" are parsed as "strictly", "trictly ", "rictly c", "ictly co", "ctly con", "tly conf", "ly confi" and so on.
- *Delimiter-based tokenization* parses a text-based payload using delimiters such as punctuation or space. For example, "strictly confidential" can be parsed using spacing to "strictly" and "confidential". The words "login.php" can be parsed into "login" and "php", where "." is the delimiter.

Transforming tokens: Our encryption algorithm is similar to the encryption mechanism of BlindBox [22] except that one of the AES computation is replaced by key-homomorphic encryption. We let $H_2 : \{0,1\}^{n_2} \times G_p \to \{0,1\}^{n_3}$ be a programmable random oracle. In particular, tokens t_i's extracted from the traffic $\mathcal{T}$ are transformed into encrypted tokens T_i's by running Algorithm 2 given below:

Note that the counter c is used to hide patterns of the encrypted tokens similar to DPIEnc of BlindBox. Because the outputs of H_1 and H_2 are deterministic, MB may learn a pattern of encrypted data if there are the same inputs. Hence, the counter $c + i$ is used to avoid this problem. Also, since the random counter c is shared and increased by one at every step, it saves the communication over-

Algorithm 2: Encryption

1. Given a traffic $\mathcal{T} = \{t_1, ..., t_{N_t}\}$, S selects a random counter c and sets $T_i = H_2(c + i, (g^{H_1(t_i)}h)^{k_{SR}})$ for all $i \in [N_t]$.
2. S sends $(c, \{T_i | i \in [N_t])$ to R (through MB).
3. S sends $\mathcal{T}$ to R via the HTTPS connection through MB.

head and allows MB to prepare the matching detection rules from $H_2(c + i, S_j)$ even before it receives encrypted tokens.

5.3 Detection

Our detection algorithm only requires N_r AES computations for each token, where N_r is the number of the detection rules negotiated in the rule setup algorithm. Therefore, the computations required for running detection algorithm are identical to those of BlindBox and PrivDPI. The following (Algorithm 3) describes the detection algorithm.

Algorithm 3: Detection

1. MB receives $(c, \{T_i | i \in [N_t]\})$ from S.
2. For all $i, i' \in [N_t]$, for all $j \in [N_r]$, MB computes $H_2(c + i, S_j)$ and compares the result with $T_{i'}$ if $i = i'$.
3. Upon finding a match, MB raises an alert.

5.4 Validation

A malicious sender may try to evade detection by creating encrypted tokens, which do not represent the actual traffic. To deal with this issue, we assume that at least one of S and R is honest. This is a realistic assumption because numerous Internet applications follow the client-server model. Our solution can be used to protect honest web servers from malicious clients or honest clients from malicious web servers. The validation process (Algorithm 4), where R is assumed to be honest, is described as follows.

6 Security Analysis

In this section, we provide security analyses on the proposed P2DPI protocol. We first review the definition of MBSE security and define rule hiding security. Based on these definitions, we prove the privacy and security of our protocol.

Algorithm 4: Validation

1. R receives $(\{t_i|i \in [N_t]\}, \{T_i|i \in [N_t]\}, c)$ from S.
2. R tokenizes and encrypts $\{t_i|i \in [N_t]\}$ into $\{T_i'|i \in [N_t]\}$ using k_{SR},
3. If $\{T_i|i \in [N_t]\} \neq \{T_i'|i \in [N_t]\}$, R warns MB.

6.1 Definitions of Middlebox Searchable Encryption and Its Security

Middlebox searchable encryption (MBSE) [22, 24] is a protocol involving interactions among RG, MB, S and R. MBSE is used to hide traffic between S and R from MB (i.e., user privacy).

Middlebox Searchable Encryption (MBSE) First, we define a (generic) MBSE scheme formally as follows.

Definition 4 (MBSE) *Middlebox searchable encryption consists of five algorithms, namely* Setup, RuleEnc, Encrypt *and* Match*:*

- Setup $[1^\lambda \rightarrow (sk, pk)]$ accepts a security parameter 1^λ and outputs an encryption key sk and a public parameter pk.
- RuleEnc $[(sk,\ r,\ pk) \rightarrow\ S)]$ accepts sk and r from a message space $\mathcal{M}$, and outputs an obfuscated rule S.
- Encrypt $[(sk, \mathcal{T}, pk) \rightarrow\ (param, \mathcal{ET})]$ accepts sk and a set of tokens $\mathcal{T} = \{t_1, ..., t_n\}$ from $\mathcal{M}$ for $n = \text{poly}(\lambda)$ and outputs encryption parameters $param$ and encrypted tokens $\mathcal{ET} = \{T_1, ..., T_n\}$.
- Match $[(param,\ \mathcal{ET},\ S) \rightarrow\ \mathcal{I}]$ accepts $param$, $\mathcal{ET}$ and S. It outputs the set of indices $\mathcal{I} = \{i|t_i = r\}$.

Correctness For any sufficiently large security parameter λ, for any polynomial number $n = poly(\lambda)$, for all $\{t_1, ..., t_n\} \in \mathcal{M}^n$, for every rule $r \in \mathcal{M}$, if S is the output of $\mathsf{RuleEnc}(sk, r, pk)$ and $(param, \mathcal{ET})$ is the output of $\mathsf{Encrypt}(sk, \mathcal{T}, pk)$ where sk and pk are the outputs of $\mathsf{Setup}(\lambda)$. $\mathsf{Match}(param,\ \mathcal{ET},\ S)$ outputs the set of index $\mathcal{I}$. For every index id_1 such that $r = t_{id_i}$ the probability that $id_1 \in I$ is 1. For every index id_2 such that $r \neq t_{id_2}$, the probability that $id_2 \in I$ is negligible.

User privacy User privacy guarantees the confidentiality of packets communicated between S and R through "MBSE security". (In other words, MBSE security guarantees the privacy of end users.) This means that the encrypted packets sent from S to R (or R to S) can be checked by MB against a set of detection rules generated by RG while the content of the packets that do not match detection rules remain confidential. The definition of the MBSE security is based on the indistinguishability game in which the adversary, having obtained

encrypted tokens of two sequences of tokens of the same length of its choice, tries to distinguish them. It is assumed that the adversary can choose any number of rules and obtains their encryption, but those rules are not included in the sets of tokens used for the encryption query. A formal definition for the MBSE security is given as follows.

Definition 5 (MBSE Security) *Let $\mathcal{A}$ be a stateful PPT adversary. Consider the following security game* $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{MBSE}}(1^{\lambda})$*:*

- Init: The challenger $\mathcal{C}$ runs Setup with a security parameter 1^{λ} to generate rule encryption keys sk and public parameter pk. It sends pk to $\mathcal{A}$.
- Query: $\mathcal{A}$ queries $\mathcal{C}$ with a set of rules $\{r_1, ..., r_q\}$. $\mathcal{C}$ invokes RuleEnc with sk and returns $\{S_1, ..., S_q\}$ to $\mathcal{A}$.
- Challenge: $\mathcal{A}$ generates two sets of tokens $\mathcal{T}_0 = \{t_{0,1}, ..., t_{0,n}\}$ and $\mathcal{T}_1 = \{t_{1,1}, ..., t_{1,n}\}$ from the message space $\mathcal{M}$ such that $\forall i \in [q], r_i \notin \mathcal{T}_0 \cup \mathcal{T}_1$. It sends both to $\mathcal{C}$. $\mathcal{C}$ randomly sets $\beta \in \{0, 1\}$ and runs Encrypt for $\mathcal{T}_\beta$ to get ($param$, $\mathcal{ET}$). $\mathcal{C}$ returns ($param$, $\mathcal{ET}$) to $\mathcal{A}$.
- Guess: $\mathcal{A}$ outputs β'. If $\beta = \beta'$, $\mathcal{A}$ wins and the game outputs 1. Otherwise, the game outputs 0.

We say that MBSE is secure if for all PPT stateful adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that

$$\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{MBSE}}(1^{\lambda}) \ = 1] = \frac{1}{2} + negl(\lambda).$$

6.2 Definitions of Rule hiding and Its Security

Rule hiding prevents S and R from knowing the current detection rules so that they cannot bypass the detection mechanism. First, we define a rule hiding scheme formally as follows:

Definition 6 (Rule Hiding Scheme) *A rule-hiding scheme consists of two algorithms, namely* Setup *and* Eval.

- Setup$[1^{\lambda} \rightarrow (sk, pp)]$ *accepts a security parameter 1^{λ}. It selects a rule hiding key sk in Z_p and publishes the public parameters pp for* Eval.
- Eval *$[(sk, r) \rightarrow R\ )]$ accepts a rule hiding key sk and a rule r from the message space $\mathcal{R}$. It outputs the obfuscated rule R of r.*

Our rule hiding security model follows the typical indistinguishability game: An end user, acting as an adversary, is allowed to know some hidden rules in Query phase, but cannot get any information about the target hidden rules given in Challenge phase. (In our protocol, the end user cannot query any rules to MB, but the user may observe the behavior of MB and may get some information about the (censored) rules. Our security model reflects this by allowing queries from the adversary.) As follows in the formal definition of the rule hiding security.

Definition 7 (Rule Hiding Security) *Let $\mathcal{A}$ be a stateful PPT adversary. Consider the following security game* $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RH}}(1^{\lambda})$ *consisting of four steps (*Init, Query, Challenge, Guess*):*

- Init: The challenger $\mathcal{C}$ runs Setup with a security parameter 1^{λ} to generate a rule hiding key sk.
- Query: $\mathcal{A}$ queries $\mathcal{C}$ for rules r_i, where $i = [n-1]$. For each r_i, $\mathcal{C}$ returns the obfuscated R_i to $\mathcal{A}$.
- Challenge: $\mathcal{A}$ generates two rules $r_{0,n}$ and $r_{1,n}$ such that $r_{0,n}$ and $r_{1,n}$ were never queried before and sends them to $\mathcal{C}$. $\mathcal{C}$ randomly selects $\beta \in \{0,1\}$. $\mathcal{C}$ returns $R_{\beta,n}$ to $\mathcal{A}$.
- Guess: $\mathcal{A}$ outputs $\beta' \in \{0,1\}$, if $\beta = \beta'$, $\mathcal{A}$ wins and the game outputs 1. Otherwise, it outputs 0.

We say that P2DPI holds a rule hiding security if for all PPT stateful $\mathcal{A}$, there exists a negligible function $negl$ such that:

$$\Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{RH}}(1^{\lambda}) = 1] = \frac{1}{2} + negl(\lambda).$$

6.3 Proof of MBSE Security

To prove the MBSE security (user privacy) of the P2DPI protocol, we extract an MBSE scheme from it, which we denote by MBSE_{P2DPI}, as follows.

Let $H_2 : \{0,1\}^{n_2} \times G_p \rightarrow \{0,1\}^{n_3}$ be a programmable random oracle with salts, which takes n_2 bits as a salt and a group element and outputting n_3 bits. MBSE_{P2DPI} is determined by the following algorithms:

- $\mathsf{Setup}[1^{\lambda} \rightarrow (k_{SR}, pk)]$ accepts a security parameter 1^{λ}. It selects encryption key k_{SR} in Z_p. It publishes the description of a pseudorandom function F_{KH} and a random oracle H_2 as public parameters pk.
- $\mathsf{RuleEnc}[(k_{SR}, r, pk) \rightarrow S]$ accepts (k_{SR}, r, pk) and calculates a session rule S where
$$S = F_{\mathrm{KH}}(k_{SR}, x_i)$$
- Encrypt $[(k_{SR}, \{t_1, ..., t_n\}, pk) \rightarrow (param, \{T_1, ..., T_n\})]$ accepts k_{SR} , pk and a set of tokens $\{t_1, ..., t_n\}$ from the message space $\mathcal{M}$ for $n = \mathrm{poly}(\lambda)$ and it outputs a $param = c_0$ and the encrypted tokens $\{T_1, ..., T_n\}$, where
$$T_i := H_2(c_0 + i, F_{\mathrm{KH}}(k_{SR}, x_i)).$$
- Match $[(param, \mathcal{ET}, S) \rightarrow \mathcal{I}]$ accepts the initial counter c_0 from $param$, a session rule S and encrypted tokens $\{T_i | i \in [n]\}$ and outputs the set of indexes
$$\mathcal{I} = \{i | S = T_i, 1 \leq i \leq n\}.$$

We show that P2DPI provides *user privacy* by proving the MBSE security of the above MBSE_{P2DPI} scheme:

Theorem 1. *(*MBSE Security of MBSE_{P2DPI}*) For a key-homomorphic PRF $F_{\text{KH}}(k_{SR}, x_i)$ (in which H_1 is assumed as a random oracle), the $MBSE_{P2DPI}$ scheme is MBSE-secure assuming that H_2 is a random oracle.*

$$\Pr[\mathsf{Exp}^{\mathsf{MBSE}}_{\mathcal{A},P2DPI}(1^\lambda) = 1] \leq \frac{1}{2} + negl(\lambda)$$

Proof. A programmable random oracle H_2 and a pseudorandom function $F_{KH}(k_{SR}, x_i)$. We let $T_t = \{t_{t,i} | t_{t,i} \in T_0 \cup T_1\}$. We can prove that the security of MBSE using the following games:

- Game_0^{MBSE}: This is the original game defined in Definition 6.2.
- Game_1^{MBSE}: This is identical with Game_0^{MBSE} except that for all $x_i \in \{r_1, \ldots r_q\} \cup \mathcal{T}_0 \cup \mathcal{T}_1$, $F_{KH}(k_{SR}, x_i)$ replaced to a random value RV_i with the restriction that it preserves equality (i.e. if $x_i = x_j (\in X)$, then $RV_i = RV_j$ $(\in \mathbb{G}_p)$).
- Game_2^{MBSE}: This is identical with Game_1^{MBSE} except that for all $x_i \in \mathcal{T}_0 \cup \mathcal{T}_1$, $H_2(c_0 + i, F_{KH}(k_{SR}, x_i))$ replaced to random values RV_i' with the restriction that it preserves equality on i and x_i.

Game_0^{MBSE} and Game_1^{MBSE} is identical because the outputs of $F_{KH}(k_{SR}, \cdot)$ are uniformly distributed and preserve equality. Because the adversary does not have k_{SR}, it cannot compute $F_{\text{KH}}(k_{SR}, y) = (g^{H_1(y)}h)^{k_{SR}}$ for $y \notin \{r_1, \ldots r_q\} \cup \mathcal{T}_0 \cup \mathcal{T}_1$ to test whether the returns are random or pseudo-random.

Game_1^{MBSE} and Game_2^{MBSE} is identical because H_2 is a random oracle. Because of the incremental counter, the adversary receives identical random values if $t_{0,i} = t_{1,j}$ and $i = j$ for $t_{0,i} \in T_0$ and $t_{1,j} \in T_0$. Otherwise, it always receives different random values.

6.4 Proof of Rule Hiding Security

To prove the rule hiding security of the P2DPI protocol, we extract a rule hiding scheme from the protocol, which we denote by RH_{P2DPI}, as follows.

Let $H_1 : \{0,1\}^{n_1} \to Z_p$ be a programmable random oracle taking n_1 bits as an input and outputting an element in Z_p. RH_{P2DPI} is determined by the following algorithms:

- $\mathsf{Setup}[1^\lambda \to (k_{MB}, pp)]$ accepts a security parameter 1^λ. It selects an obfuscation key k_{MB} in Z_p. It publishes the description of a function H_1 and a generator g as public parameters pp.
- Eval $[(k_{MB}, r, pp) \to R]$ accepts k_{MB}, pp and a rule r from the message space $\mathcal{R}$. It outputs the obfuscated rule R where

$$R := (g^{H_1(r)}h)^{k_{MB}}.$$

We prove that rule hiding security holds for our P2DPI protocol by showing that the rule hiding security of the above is reduced to the PRF security.

Theorem 2. *(*Rule hiding of P2DPI*) Suppose DDH Assumption holds. Then, RH_{P2DPI} satisfies the rule hiding security or*

$$\Pr[\mathsf{Exp}^{\mathsf{RH}}_{\mathcal{A},P2DPI}(1^\lambda) = 1] \leq \frac{1}{2} + negl_{DDH}(\lambda).$$

Proof. Note that the function used to create R in the above RH_{P2DPI} is the key-homomorphic function $F_{\mathrm{KH}}(k, x) = (g^{H_1(x)}h)^k$, where k is set as k_{MB}. Hence, we can simulate the rule-hiding game $\mathsf{Exp}^{\mathsf{RH}}_{\mathcal{A},P2DPI}(1^\lambda)$ for $\mathcal{A}$ using a PRF adversary $\mathcal{B}$ for F_{KH}: Whenever $\mathcal{A}$ queries rules r_i's in Query phase, $\mathcal{B}$ forwards them to its oracle (which is either a random function or a prf) to get the answers and forwards them back to $\mathcal{A}$. When $\mathcal{A}$ queries rules $(r_{0,n}, r_{1,n})$ in Guess phase, $\mathcal{B}$ selects $b \in \{0, 1\}$ at random and forwards $r_{b,n}$ its oracle to get the target rule and forwards it back to $\mathcal{A}$. Thus, we obtain $\Pr[\mathsf{Exp}^{\mathsf{RH}}_{\mathcal{A},P2DPI}(1^\lambda) = 1] \leq \Pr[\mathsf{Exp}^{\mathsf{PRF}}_{\mathcal{B}}(1^\lambda) = 1]$. Since F_{KH} is shown to be secure PRF according to [16] and [3] under the DDH assumption in the random oracle model, we have $\Pr[\mathsf{Exp}^{\mathsf{PRF}}_{\mathcal{B}}(1^\lambda) = 1] \leq \frac{1}{2} + negl_{DDH}(\lambda)$, obtaining the bound in the theorem statement.

The proof implies that the adversary (i.e., the end user) cannot distinguish an obfuscated rule R from a random value even if it knows some other obfuscated rules $R_i = (g^{H_1(r_i)}h)^{k_{KB}}$ for $i = 1, \ldots, q$, where q is the number of rule queries. Therefore, *rule hiding* security holds.

7 Performance Evaluation

We implemented Sender (S), Receiver (R), Rule Generator (RG) and MiddleBox (MB) of our P2DPI protocol and evaluated their performance. The performance analysis of our P2DPI was conducted on a machine whose base OS is Ubuntu 18.04LTS, equipped with Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 3.60GHz with 16.0 GB memory. We used the C programming language to implement P2DPI. For the big number and elliptic curve operations, we used the OpenSSL library [19]. We also used the Python ECDSA package [4] for a digital signature. The followings are implementation descriptions for each P2DPI component.

S and R: We implemented an SSL client as the sender (S) using the OpenSSL library. We did not analyze the Validation algorithm in the receiver because it is identical to the Encryption algorithm of the sender.

RG: In our P2DPI, we simply used Snort Emerging Threats to construct detection rules r_i. The rule generator RG uses the OpenSSL library to compute R_i from r_i and the Python ECDSA package to sign R_i to produce $sig(R_i)$.

MB: We implemented the middlebox MB also using the OpenSSL library to realize the cryptographic operations that are required for the Setup and Detection algorithms.

7.1 Experiment Environments

In this section, we compare our performance results with those of PrivDPI. According to [18], PrivDPI was implemented using Python. However, we used the *C* programming language to implement our P2DPI. Therefore, for the fair comparison which does not depend on the performance variations from the underlying programming languages, we implemented PrivDPI using *C*.

Our P2DPI requires hash and elliptic curve operations. We used the benchmark command of OpenSSL to compare the performance of several candidate algorithms. We implemented hash functions H_1 and H_2 using the AES algorithm to model programmable random oracles. More specifically, H_1 and H_2 are constructed as follows:

$$H_1(x) := AES128_x(Salt),\ H_2(y,h) := AES128_{h \bmod 2^{128}}(y),$$

where *Salt* in $H_1(\cdot)$ is a salt value that is fixed for the implementation; h is a coordinate of a group element over the prime field *prime256v1*; and x and y are from the domains of keywords and counters.

We employed AES since it is fast as the most modern CPUs provide a hardware accelerator for it, such as AES-NI instructions. (We utilized EVP_CIPHER object in the OpenSSL library in order to accelerate the encryption by the hardware accelerator.) We used CBC mode for the AES operation.

For elliptic curve operations, such as scalar multiplications, we used the NIST P256 curve, which was one of the fastest options we can use with the OpenSSL library.

7.2 Rule Setup

In P2DPI, RG, MB and S/R interact with each other to exchange the detection rules, which are created by RG, in an obfuscated format to prepare for the packet inspection among S, R and MB. We select 2,000 rules from "emerging-trojan.rules", which contains 5,565 keywords and each of which is tokenized by the size of 8 bytes from Snort emerging threats rule. The number of keywords for each rule is given in Table 1. Rule Setup can be divided into two parts. One is rule preparation permitted by RG, which is executed only when there is an update of detection rules, which probably once in a few days. The other is session rule exchange between MB and S/R, which is performed whenever there is a new connection request between a sender (S) and a receiver (R). Note that the session rule exchange is more critical to the system's usability as it impacts the initial handshake to establish a secure communication channel between the users (S and R). We separately measure the rule preparation and session rule exchange times to estimate the computational overhead of RG and a delay in session establishment.

P2DPI shows the security improvement mentioned above without sacrificing performance. The computational overhead imposed on Rule Setup is almost identical. It is even slightly reduced in P2DPI compared to PrivDPI. We compare

Table 1. Rule Sizes

# of Rules	1	500	1000	1500	2000
# of Keywords	8	1366	2727	4156	5565

the processing times of rule preparation and session rule exchange in Tables 2 and 3. Because the times taking for signing and verification overwhelm in those computations and blind the difference between P2DPI and PrivDPI, we also measured the overhead without signature schemes to present the difference. The times estimated without the signature scheme are denoted using "(w/o Sign)".

As shown in Table 2, with 1,000 rules containing 2,727 keywords, the rule preparation time of PrivDPI took about 704 ms with signature generation in the estimation, and 52 ms without signature generation and these are respectively reduced slightly to 698 ms and 46 ms in our P2DPI. The reduction is caused by the difference in the rule generation mechanism. Compared to P2DPI, PrivDPI needs an additional randomization parameter, s_i, for each keyword and stores it to deliver it to MB. This requirement causes this difference, although it is not significant.

The session rule exchange time is presented in Table 3. The session rule exchange time of P2DPI is still slightly less than that of PrivDPI as it needs less computation on elliptic curves. With 1,000 rules, the session rule exchange process of P2DPI takes 2.84 seconds with signature verification and 0.30 seconds without signature verification. In PrivDPI, the same process takes 2.87 seconds and 0.33 seconds, which are slightly slower than P2DPI.

Table 2. Rule Generation Time by RG (ms)

# of Rules	**P2DPI**	**P2DPI** (w/o Sign)	PrivDPI	PrivDPI (w/o Sign)
1	**1.631**	**0.378**	1.915	0.662
500	**351.780**	**23.073**	354.765	26.058
1000	**698.084**	**45.921**	704.071	51.908
1500	**1038.264**	**70.417**	1043.622	75.776
2000	**1385.290**	**93.604**	1393.663	101.977

Bandwidth: Compared with the plain TLS allowing no inspection, P2DPI consumes more bandwidth as it requires exchanging obfuscated detection rules in the Rule Setup. We present the additional bandwidth required for Rules Setup between MB and S (or MB and R) in Table 4. It should be noted that this bandwidth consumption is almost identical to that of PrivDPI except that P2DPI requires one fewer group elements per connection in the Rule Setup compared with PrivDPI.

Table 3. Session Rule Exchange Timebetween MB and S/R (ms)

# of Rules	**P2DPI**	**P2DPI** (w/o Sign)	PrivDPI	PrivDPI (w/o Sign)
1	**4.270**	**1.426**	4.798	1.954
500	**1419.184**	**148.895**	1438.990	168.701
1000	**2835.729**	**296.774**	2868.596	329.641
1500	**4317.805**	**449.506**	4362.262	493.963
2000	**5741.694**	**601.072**	5804.661	664.039

Table 4. Bandwidth (KBytes)

# of Rules	1	500	1000	1500	2000
Bandwidth	1.2	206.9	413.1	628.0	840.4
Bandwidth (w/o Sign)	1.1	174.9	349.1	532.0	712.4

7.3 Encryption

P2DPI provides faster encryption compared with PrivDPI. This is because P2DPI has less elliptic curve operations to encrypt each token. Similar to the rule setup algorithm, P2DPI needs only one scalar multiplication although PrivDPI needs one scalar multiplication and one point addition. Ours may be still slower than the encryption of BlindBox, which does not require any elliptic curve operation. However, the performance gap is reduced when we compare the encryption speed of P2DPI with that of PrivDPI. More precisely, the encryption of P2DPI is about 3.5 times faster than PrivDPI. The efficiency improvement originates from the way both implementations of P2DPI and PrivDPI use the EC_POINT_MUL operation in the OpenSSL library, which is to realize the scalar multiplication over elliptic curve: In P2DPI, S and R apply scalar multiplications to the fixed generator g, while in PrivDPI, S and R apply them to an arbitrary point g^{α}. On the other hand, in PrivDPI, S and R encrypt a token t_i to $g^{k \cdot \alpha \cdot t_i + k^2}$ to enable MB to match it with the encrypted rules (See preprocessing protocol of PrivDPI in Section 3). Because S and R know the shared key k and the token t_i, but they do not know α, they need to receive the public parameter $A = g^{\alpha}$ from MB right after the connection is established. Then, they will start to encrypt all tokens by computing $A^{k \cdot t_i} \cdot g^{k^2}$.

The distinction stems from the above difference between the design of P2DPI and that of PrivDPI. To be precise, encrypting tokens requires a scalar multiplication over the generator g to $g^{H_1(t_i) \cdot k_{SR}}$ and add $h^{k_{SR}}$ to compute $g^{H_1(t_i) \cdot k_{SR}} \cdot h^{k_{SR}}$ in P2DPI, while that requires a scalar multiplication over the arbitrary point $A = g^{\alpha}$ in PrivDPI. Our micro-analysis showed that the computation of the former case is 5.5 times faster than that of the latter case. In particular, the scalar multiplication over a generator takes around 8 μs, while that over an arbitrary point takes 44 μs in the well-known OpenSSL library. Therefore,

performing scalar multiplications over the fixed generator, g, turns out to be a significant improvement in the encryption algorithm of P2DPI over PrivDPI.

Table 5. Encryption Time (ms)

# of tokens	**P2DPI**	PrivDPI
1k tokens	**15.227**	55.229
2k tokens	**33.586**	109.434
3k tokens	**46.945**	163.826
4k tokens	**62.480**	218.338
5k tokens	**76.122**	272.535

We estimated loading time of P2DPI for some popular websites[6], which are BBC (British Broadcasting Corporation, 4.4 MB), BOA (Bank of America, 6.5 MB) and NYT (the New York Times, 10.5 MB). As shown in Figure 2, the loading time for those pages is not prohibitively slow, which ranges from 1.1 s to 2.3 s with 1k rules and from 1.4 s to 2.6 s with 2k rules. This result shows that P2DPI provides much faster loading time compared to that of PrivDPI, which ranges from 3.3 s to 7.5 s with 1k rules and from 3.6 s to 7.8 s with 2k rules. It should be noted that signature validation time is not included in the loading time as the validations are only needed when S and R receive updated or new rules; also, webpages are tokenized by 8 bytes on average. The estimation shows that P2DPI is more practical, compared with PrivDPI and BlindBox. We excluded BlindBox in the comparison as it obviously requires several tens of seconds only for the connection setup according to [18, 22].

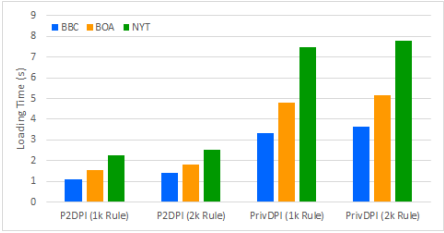

Fig. 2. Loading Times of Popular Websites (s)

[6] We assume that image files (e.g., `*.jpg` and `*.png`) and video streams (e.g., `*.mp4`) are not monitored. The sizes of webpages referred to here do not include those of image files or streaming videos.

7.4 Detection

We note that there is no theoretical difference in detection algorithms among our P2DPI, BlindBox and PrivDPI. Like BlindBox and PrivDPI, P2DPI provides the fastest detection over encrypted tokens in the literature. The detection depends on both the number of detection rules and the number of tokens. The results are shown in Table 6.

Table 6. P2DPI Detection Time (ms)

The number of tokens	The number of rules (# of Keywords)	
	1 rule (8)	1000 rules (2727)
1 token	**0.037**	**0.807**
1k tokens	**4.229**	**490.541**

8 Related Work

BlindBox: Sherry et al. [22] introduced the first practical DPI solution over encrypted traffic. While BlindBox is capable of fast packet inspection, it has a few drawbacks. BlindBox requires detection rules to be encrypted by a secret key, which is exchanged between a sender and a receiver (i.e. a user/client and a server) but cannot be disclosed to the middlebox as our P2DPI does. To achieve this, BlindBox uses oblivious transfer [15, 2] and the Yao's garbling scheme [26], which are computationally expensive and cause substantial latency when the connection is initiated. For example, BlindBox requires 97 seconds to process 3,000 detection rules, which contain around 10,000 keywords. BlindBox seems to be not suitable for applications requiring frequent re-connections such as web browsing, where a new TLS/SSL connection is established for each HTTP request.

BlindIDS: BlindIDS from Canard et al. [5] does not require any additional computation to set-up a connection. It also provides market-compliance. BlindIDS randomizes the detection rules, i.e. it makes them indistinguishable from random values of the same length. However, BlindIDS is based on a searchable encryption scheme constructed from bilinear maps (i.e. pairing). Meaning that actual inspection time is many orders of magnitude slower than BlindBox. For example, an inspection of one token using one rule becomes more than 34,000 times slower. This makes BlindIDS impractical.

Outsourced Middleboxes: Middlebox functions can be outsourced to a cloud [13, 8], which might be outside of organization networks. This system architecture makes BlindBox more practical since sessions between gateways last longer.

However, this does not overcome the fundamental drawback of BlindBox for applications requiring frequent re-connections.

Other Approaches: Yan et al.'s scheme prepares detection rules using a randomization algorithm [27]. Their scheme does not suffer from a long delay at the setup stage. However, it requires the "admin server" to create a randomization key for senders and receivers, and deliver the randomized rules to middleboxes. This system leaks information about users' communication if middleboxes and the admin server collude. Similarly, a system proposed in [9], termed "SGX-box", facilitates the middlebox with special hardware to get the user's session keys in real time. However, the problems of this approach are that the middlebox should be always online to provide the inspection service, and a secure channel between the user and the middlebox must be established.

Multi-context TLS protocol (mcTLS) [17] was introduced to alleviate the concern of data privacy. It enforces middlebox to obtain approvals from both a client and a server for the inspection. However, it is still based on the MITM approach, which requires to share TLS session keys with middlebox. It also requires more computational and communication overheads for the inspection.

The deep learning technology is used to classify encrypted traffic [14]. However, it only classifies the traffic using its protocol and some other characteristics. It cannot be applicable to check the contents of the traffic.

Expressiveness of detection rules: P2DPI uses exact keyword matching for detection. In other words, it matches keywords (i.e., detection rules) against the traffic and raises alerts if they are matched. Full IDS systems require more expressive rules such as regular expressions or scripts rather than the keywords only. According to Sherry et al. [22], a single keyword exact matching can cover 1.6 - 5 % of the policies that general HTTP IDS applications require. If multiple keywords are used for exact matching, the coverage increases to 29 - 67 % depending on the HTTP IDS applications. The regular expressions and scripts can be covered by first applying exact keyword matching, then partially decrypting the traffic where keywords are matched to apply the rules written in regular expressions or scripts to the traffic. However, to the best of our knowledge, there has been no efficient algorithm to support the expressive rules without decryption for the scenario P2DPI, BlindBox and PrivDPI support.

9 Conclusion

In this paper, we introduced a new deep packet inspection protocol, P2DPI. We discussed that the existing solutions for the deep packet inspection exhibit either impractical performance on initiating encryption channel [22] or weaker security, which cannot meet the security level that the deep packet inspection (DPI) system commonly requires. Notably, we showed that the user privacy of PrivDPI [18] cannot be guaranteed against the rule generator. P2DPI resolves the security problem and achieves the required security level of the DPI protocol, originally suggested in [22]. Besides, from the efficiency perspective, P2DPI outperforms PrivDPI, which is the most efficient DPI protocols in the literature, to

the best of our knowledge. P2DPI additionally improves the packet encryption time about 3.5 times. Our result is supported by the formal security analysis and the performance analysis based on our implementation results.

References

1. Https encryption on the web. https://transparencyreport.google.com/https/overview?hl=en. Accessed: 2019-07-28.
2. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 2013, Berlin, Germany, November 4-8, 2013*, pages 535–548. ACM, 2013.
3. Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *CRYPTO 2013, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, 2013.
4. Brian Warner. Pure-python ecdsa, 2020. accessed 2020-05-21.
5. Sébastien Canard, Aïda Diop, Nizar Kheir, Marie Paindavoine, and Mohamed Sabt. Blindids: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic. In *ACM AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 561–574. ACM, 2017.
6. Colin Dixon, Hardeep Uppal, Vjekoslav Brajkovic, Dane Brandon, Thomas E. Anderson, and Arvind Krishnamurthy. ETTM: A scalable fault tolerant network manager. In David G. Andersen and Sylvia Ratnasamy, editors, *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*. USENIX Association, 2011.
7. Thomas Fuhr and Pascal Paillier. Decryptable searchable encryption. In *ProvSec 2007, Wollongong, Australia, November 1-2, 2007, Proceedings*, volume 4784 of *LNCS*, pages 228–236. Springer, 2007.
8. Yu Guo, Cong Wang, and Xiaohua Jia. Enabling secure and dynamic deep packet inspection in outsourced middleboxes. In *SCCAsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 49–55. ACM, 2018.
9. Juhyeng Han, Seong Min Kim, Jaehyeong Ha, and Dongsu Han. Sgx-box: Enabling visibility on encrypted traffic using a secure middlebox module. In *APNet 2017, Hong Kong, China, August 3-4, 2017*, pages 99–105. ACM, 2017.
10. Lin-Shung Huang, Alex Rice, Erling Ellingsen, and Collin Jackson. Analyzing forged SSL certificates in the wild. In *IEEE S&P 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 83–97. IEEE Computer Society, 2014.
11. Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012, Raleigh, NC, USA, October 16-18, 2012*, pages 965–976. ACM, 2012.
12. Jongkil Kim, Seyit Camtepe, Joonsang Baek, Willy Susilo, Josef Pieprzyk, and Surya Nepal. P2DPI: practical and privacy-preserving deep packet inspection. In Jiannong Cao, Man Ho Au, Zhiqiang Lin, and Moti Yung, editors, *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, pages 135–146. ACM, 2021.
13. Chang Lan, Justine Sherry, Raluca Ada Popa, Sylvia Ratnasamy, and Zhi Liu. Embark: Securely outsourcing middleboxes to the cloud. In Katerina J. Argyraki

and Rebecca Isaacs, editors, *USENIX NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*, pages 255–273. USENIX Association, 2016.
14. Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: a novel approach for encrypted traffic classification using deep learning. *Soft Comput.*, 24(3):1999–2012, 2020.
15. Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In Michael J. Wiener, editor, *CRYPTO1999, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *LNCS*, pages 573–590. Springer, 1999.
16. Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In *EUROCRYPT 1999, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *LNCS*, pages 327–346. Springer, 1999.
17. David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodríguez Rodríguez, and Peter Steenkiste. Multi-context TLS (mctls): Enabling secure in-network functionality in TLS. In Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye, editors, *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pages 199–212. ACM, 2015.
18. Jianting Ning, Geong Sen Poh, Jia-Ch'ng Loh, Jason Chia, and Ee-Chien Chang. Privdpi: Privacy-preserving encrypted traffic inspection with reusable obfuscated rules. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *2019 ACMCCS 2019, London, UK, November 11-15, 2019*, pages 1657–1670. ACM, 2019.
19. OpenSSL Software Foundation. Openssl, 2020. accessed 2020-02-08.
20. Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.
21. Antonio Cuadra Sánchez and Javier Aracil. A novel blind traffic analysis technique for detection of *WhatsApp* voip calls. *Int. Journal of Network Management*, 27(2), 2017.
22. Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pages 213–226. ACM, 2015.
23. Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. *IACR Cryptol. ePrint Arch.*, 2015:264, 2015.
24. Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE S&P 2000, Berkeley, California, USA, May 14-17, 2000*, pages 44–55. IEEE Computer Society, 2000.
25. Verison. Insider threat report. https://enterprise.verizon.com/resources/reports/insider-threat-report.pdf, 2020. accessed 2020-11-08.
26. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.
27. Xingliang Yuan, Xinyu Wang, Jianxiong Lin, and Cong Wang. Privacy-preserving deep packet inspection in outsourced middleboxes. In *IEEE INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, pages 1–9. IEEE, 2016.