Towards Efficient Privacy-Preserving Deep Packet Inspection

Weicheng Wang[1], Hyunwoo Lee[2], Yan Huang[3], Elisa Bertino[1], and Ninghui Li[1]

[1] Purdue University, West Lafayette IN 47907, USA
{wang3623,bertino,ninghui}@purdue.edu
[2] KENTECH, Naju-si, Jeonnam 58330, Republic of Korea
hwlee@kentech.ac.kr
[3] Indiana University, Bloomington IN, 47405, USA
yh33@indiana.edu

Abstract. Secure Keyword-based Deep Packet Inspection (KDPI) allows a middlebox and a network sender (or receiver) to collaborate in fighting spams, viruses, and intrusions without fully trusting each other on the secret keyword list and encrypted traffic. Existing KDPI proposals have a heavy-weighted initialization phase, but also require dramatic changes to existing encryption methods used to the original network traffic during the inspection phase. In this work, we propose novel KDPI schemes MT-DPI, CE-DPI, BH-DPI which offer highly competitive performance in initialization and guarantee keyword integrity against malicious middlebox. Moreover, our methods work readily with AES-based encryption schemes that are already widely deployed and well-supported by AES-NI. We show that our KDPI schemes can be integrated with TLS, adding marginal overhead.

Keywords: MPC · DPI · BlindBox · Garbled Circuit · OT

1 Introduction

Deep packet inspection (DPI) is a widely used security technique for detecting intrusions, scanning for malware, and preventing data exfiltration [54, 8]. DPI relies on a middlebox that sits between two communication endpoints to examine the payloads of network traffic to improve security [40, 11]. Many DPI solutions, e.g., OpenDPI [29] and nDPI [14], have been adopted by enterprises. The global DPI market size is projected to reach 16.6 billion USD by 2026, up from 4.6 billion USD in 2020, at a yearly growth rate of 25.0% during 2021-2026 [52].

Since Transport Layer Security (TLS) [16] was introduced to secure Internet communications, its usage has been growing [41, 20]. At the time of writing, more than 90% of HTTP traffic from Google products is encrypted with TLS, according to Google Transparency Report [22]. However, the end-to-end security provided by TLS makes it challenging for a middlebox to perform DPI. Some proposals such as SplitTLS [28] used a TLS interception scheme to allow a middlebox to access plaintext traffic, which, unfortunately, turns out introducing

vulnerabilities [10, 17, 48, 56], and compromise the privacy of TLS [55, 38]. This situation prompted the research community to consider alternative solutions [42].

In 2015, Sherry et al. [53] proposed BlindBox, a keyword-based DPI protocol, which enables the middlebox to detect occurrences of preset keywords in the encrypted traffic, while assuring that the middlebox does not learn other information about the encrypted traffic and that the communicating endpoints do not learn the keywords. The initialization phase of their protocol heavily relied on the garbled circuit protocol to securely generate tokens for each keyword. Their approach, however, has three serious limitations: (1) It only considered semi-honest middleboxes and did not offer integrity of the keywords against malicious middleboxes. Hence, although legitimate keywords (also known as rules) should only be decided by a third party called Rule Generator (RG), BlindBox did not prevent a corrupted middlebox from replacing legitimate keywords with any keywords of an adversary's choice to compromise traffic privacy. (2) As their experiments showed, BlindBox's initialization phase is rather heavy-weighted, requiring gigabytes of communication overhead [53], even without guaranteeing keyword integrity. (3) Their traffic inspection phase is very expensive, since it requires invoking the AES cipher a large number of times each with a different AES key, making it hard to leverage efficient pipelining of AES-NI instructions.

Therefore, we asked, "Can we have KDPI protocols that require minimal modification to the already wide-deployed symmetric-key encryption schemes while offering keyword integrity and performing significantly better than BlindBox?" In this regard, we study new constructions of KDPI and provide positive answers to the question above.

Contributions Aiming at resolving those limitations of BlindBox, we propose three efficient keyword-based DPI protocols, MT-DPI, CE-DPI, and BH-DPI. All three protocols offer keyword integrity and can be integrated with an AES-based underlying encrypted channel with minimal change, while significantly outperforming BlindBox in both initialization and traffic inspection phases.

Our first approach, MT-DPI, is based on binary multiplicative triples that we can efficiently generate using Ferret OT [59]. The resulting protocol is an order of magnitude more efficient than BlindBox in communication cost and runs a few times faster in realistic network environments where middleboxes are deployed. A major challenge in the design of MT-DPI is keyword integrity, because the traditional secret share and multiplicative triple-based protocol cannot withstand active attacks. To address this challenge, we developed a novel use of the cut-and-choose technique for secret-share-based two-party computation protocols, which may be of independent interest. We proved our protocol is secure against actively corrupted middleboxes. Our second approach, CE-DPI, extends BlindBox with a more efficient keyword verification method using Oblivious Commitment-Based Envelope (OCBE) [35]. Compared with the keyword verification suggested by BlindBox, CE-DPI is proven secure against semi-honest middlebox but uses only 1/10 the bandwidth of BlindBox. The third approach, BH-DPI, improves BlindBox with a batched hashing verification, and uses 1/6 the bandwidth of BlindBox.

2 Background

2.1 Keyword-based DPI

We consider the problem of enabling Deep Packet Inspection (DPI) under end-to-end encryption, e.g., when TLS is used. The following parties are involved.

Sender (S) and Receiver (R). DPI is performed by a third party when the communication between S and R is encrypted. The content being sent is a sequence of packets.

Middlebox (MB). Given a set of inspection rules, MB inspects the traffic between S and R, to find out whether a packet matches any of the rules.

Rule Generator (RG). This party defines the inspection rules. An RG is trusted and specializes in creating effective rules that are useful for various DPI tasks. However, they are not trusted to inspect network traffic. In fact, they will not participate in running DPI protocols. Instead, an MB subscribes to an RG in order to obtain the rules and offer the inspection service. The role of RGs can be played by major security vendors (e.g., Symantec).

Keyword-based DPI systems. One class of DPI systems uses rules that are fixed-length **keywords**. That is, the RG creates a list of ℓ-byte keywords (where ℓ is a fixed parameter), and the goal of DPI is to determine whether a packet contains any keyword in the list.

To realize keyword-based DPI, a sender S will compute a **token** for every ℓ-byte substring (**word**) in the traffic, and send them to the MB. MB checks the token to learn whether any of them matches that of a keyword. Note each pair of neighboring ℓ-byte words has $\ell - 1$-byte overlap. So in an N-byte packet, the total number of ℓ-byte words is $N - \ell + 1$.

In KDPI, ℓ, the length of keywords, and K, the number of such keywords, are public information. Figure 1 depicts a keyword-based DPI System. A KDPI protocol has the following sub-protocols.

1. **Rule delivery.** MB and RG run a protocol so that MB obtains information about the keywords. Note that this does not necessarily mean that MB learns the actual keywords which can be RG's private business asset. Rule delivery happens infrequently, only when initializing MB's subscription or when rules need to be updated.
2. **Session initialization.** Before S establishes its connection with R, S runs an initialization protocol with MB to prepare MB to allow it to later inspect the traffic between S and R. The initialization protocol only needs to be executed once per new pair of connecting peers.
3. **Traffic inspection.** Before S sends a packet, S extracts all ℓ-byte words in the packet, and computes a token for each word (which we call *token computation*) and sends them to MB. MB inspects the tokens to determine whether each token is generated from a keyword (which we call *token inspection*).

Notations. We summarized our notations in Table 1. Here we emphasize two abstractions.

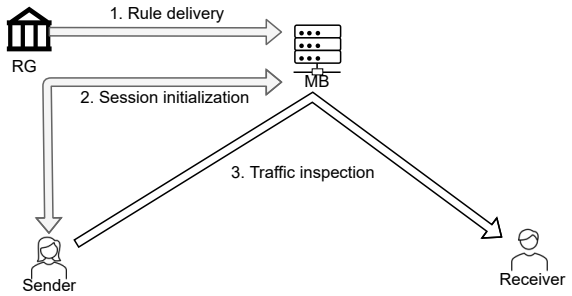

Fig. 1. A Keyword-based DPI System.

Table 1. Notation used in the paper

Symbol	Description	Symbol	Description	Symbol	Description
ℓ	length of a keyword/word	ω	a word (ℓ-byte)	w	a keyword (ℓ-byte)
W	a list of keywords	K	the number of keywords in W	$\eta(\omega)$	a handle of ω
$T(\omega)$	a token computed from ω	k	the secret key of S (or R)	ctr_ω	a counter for ω

- **Handle**: For each **word** ω, there is a corresponding handle $\eta(\omega)$. Possession of $\eta(\omega)$ enables a party to identify all occurrences of ω in the traffic. During session initialization, MB, with help from S, learns the handle $\eta(w)$ for each **keyword** $w \in W$, where W denotes all keywords determined by RG.
- **Tokens**: For each appearance of ω in the traffic, S computes a token $T(\omega)$, and sends $T(\omega)$ to MB. It is essential to ensure that different occurrences of the same word result in unlinkable tokens except to a party who knows the handle $\eta(\omega)$. One way is to use per-word counters, and set all such counters to 0 at the beginning of the session, and increment the counter for ω each time ω appears in the traffic. Then S computes tokens on the word and its corresponding counter value not to repeat the same tokens on the same word.

Traffic confidentiality. If a malicious MB colludes with RG, our goal is to ensure MB only learns the occurrences of the set of K words of their choice in the traffic from S to R. The exact leakage depends on ℓ and K. If $K \geq 2^{8\ell}$, the adversary can fully recover the traffic since the keyword list is long enough to include all possible words. In typical settings, however, $K \ll 2^{8\ell}$ (e.g., in our experiments $K = 3000$ and $\ell = 5$ or 8) so that leakage is reasonably small.

If RG and S are honest while MB is malicious, the confidentiality goal will be to ensure that MB learns only the occurrences of keywords defined by the honest RG. A malicious MB should not be able to tamper with the list of keywords and learn extra information in the traffic.

Keywords confidentiality. The list of keywords may be the valuable intellectual property of RG. Revealing these keywords could also allow attackers to circumvent the inspection mechanism. Therefore, a KDPI system should ensure that malicious S (or R) do not learn anything about the keywords beyond what could be inferred from the observable outcome of the system.

2.2 BlindBox

Sherry et al. [53] proposed BlindBox, which became a benchmark for KDPI systems.

Using AES for handles and tokens. In BlindBox, the handle for each word ω is $\eta(\omega) = \mathsf{AES}_k(\omega)$, where the key k is known to S and R, but not to MB. MB learns the handles for keywords with help from S, but cannot compute the handles for other words. The token for an occurrence of a word ω is the AES encryption using $\eta(\omega)$ as key, and the counter for ω as plaintext. Because of counters, the tokens computed from different occurrences of $\omega' \notin W$ are unlinkable by MB.

Using table lookup for inspection. At the beginning of each session, MB computes $T = \{\mathsf{AES}_{\eta(w)}(0) \mid w \in W\}$ and maintains the tokens generated from keywords in a table. (In [53], it is suggested to implement the table using a balanced binary tree.)

When MB receives a token t, it checks whether $t \in T$. That is, the check takes the form of a table lookup. When $t \in T$, this means that t is computed from a keyword w, MB can identify the keyword, update the corresponding counter, and update T by replacing $\mathsf{AES}_{\eta(w)}(0)$ with $\mathsf{AES}_{\eta(w)}(1)$. This way, MB can maintain counters for $w \in W$ in synchronization with S, so long as it processes all the tokens in order.

Token computation overhead. For S to compute a token, it needs to find the counter corresponding to the word, and perform 2 AES operations using two different keys.

Token transmission overhead. In order to match keywords that appear in arbitrary positions of the payload, S needs to use a sliding window to extract words to compute tokens. The number of tokens is thus about the same as the length of the payload in bytes. That is, for every byte in the payload, S needs to send a token to MB. If each transmitted token is 16 bytes (i.e., 128 bits), then the extra communication overhead is 16x the payload.

This overhead can be reduced by using truncated tokens. For example, one can only use the first 5-bytes of the AES encryption to make a 5-byte token. This results in reducing the communication overhead from 16x to 5x. See 3.4 for more details.

Using garbled circuits to hide rules. To ensure that MB does not learn the handles for words not in W, and S does not learn W, BlindBox uses a secure two-party computation protocol. RG generates a signature for each $w \in W$ using its own public key. MB and S run Yao's Garbled Circuit protocol where MB provides w and RG's signature over w, and S provides k.

Here the garbled circuit first verifies that the signature on w is valid, and if so, gives the handle $\eta(w) = \mathsf{AES}_k(w)$ to MB. The usage of garbled circuits introduces

significant overhead in terms of both time and bandwidth. This overhead is especially large if signature verification needs to be part of the circuit.

Drawbacks of BlindBox. As we discussed in the previous sections, BlindBox is less efficient in the token initialization phases due to the high bandwidth requirements for the garbled circuit. The initialization phase requires roughly 4.3GB bandwidth between S and MB which makes the BlindBox hard to be practical. Another drawback for BlindBox is the lacking of keyword verification. The threat model of BlindBox assumes that MB is semi-honest but in practice MB may replace the keywords to other sets of words without being detected. Though the paper suggests using a hash to verify each keyword before initialization, it did not clearly describe the steps.

2.3 Secure Two-Party Computation Protocols

Given a Boolean circuit, cryptographically secure computation protocols allow two mutually-distrustful parties to compute the circuit over their secret inputs. *Garbled Circuit* and *Additive Secret-Sharing based* protocols represent two approaches to cryptographic secure computation. The garbled circuit approach features a small constant round of interaction but requires a large amount of network bandwidth, typically at least 32 bytes per binary AND gate. In contrast, additive secret-sharing based protocols cost a linear number of rounds in the depth of the circuit but little bandwidth, about 2 bits per binary AND gate. We refer the readers to the literature [60, 37, 18] for the garbled circuit technique, but describe Beaver's protocol [5] as a representative secret-sharing based approach since it is used in our main protocol.

Beaver's Circuit Randomization Protocol Let C be a binary circuit consisting of XOR and AND gates. Two parties P_1 with input x and P_2 with input y want to compute $C(x, y)$ without leaking extra information about x, y. Beaver's protocol process the circuit in a topological order:

- For every initial input associated with secret bit v owned by P_i: P_i divides v into two uniform random bits v_1, v_2 where $v_1 \oplus v_2 = v$ and distribute the shares such that P_i holds v_i.
- For every binary gate with secret input bits u, v: let the secret shares of u be u_1, u_2 and that of v be v_1, v_2 where P_1 holds u_1, v_1 while P_2 holds u_2, v_2.
  - If it is an XOR gate, to derive the shares of the secret output bit $z = u \oplus v$, P_1 sets $z_1 \coloneqq u_1 \oplus v_1$ and P_2 sets $z_2 \coloneqq u_2 \oplus v_2$.
  - If it is an AND gate, to derive the shares of the secret output bit $z = u \cdot v$,
    1. P_1 and P_2 generate a multiplicative triple (see Section 2.3 for the triple generation protocol) so that P_1 obtains random bits a_1, b_1, c_1 while P_2 obtains random bits a_2, b_2, c_2 such that $(a_1 \oplus a_2) \cdot (b_1 \oplus b_2) = c_1 \oplus c_2$.
    2. P_1 and P_2 compute u', v' where $u' = u \oplus a$ and $v' = v \oplus b$, by revealing $u_1 \oplus a_1$, $u_2 \oplus a_2$, $v_1 \oplus b_1$, $v_2 \oplus b_2$.
    3. P_1 sets $z_1 \coloneqq u' \cdot v' \oplus u' \cdot b_1 \oplus v' \cdot a_1 \oplus c_1$. P_2 sets $z_2 \coloneqq u' \cdot b_2 \oplus v' \cdot a_2 \oplus c_2$.

– To reveal the secret value v on an output wire of the whole circuit, P_1 and P_2 simply exchange their respective secret shares of v and each party locally xor the shares.

For correctness, it is easy to verify that $z_1 \oplus z_2 = u' \cdot v' \oplus u' \cdot (b_1 \oplus b_2) \oplus v' \cdot (a_1 \oplus a_2) \oplus (c_1 \oplus c_2) = u' \cdot v' \oplus u' \cdot b \oplus v' \cdot a \oplus a \cdot b = (u' \oplus a) \cdot (v' \oplus b) = u \cdot v$. For confidentiality, we note that all exchanged bits are uniform random and independent of each party's secrets.

Generating Multiplicative Triples Multiplicative triples used above for securely computing the ANDs are typically computed in batches using Random Oblivious Transfers (ROT). ROT is a two-party cryptographic functionality that, once invoked, will generate two uniform random strings x_1, x_2 and one uniform random bit b, then sends (x_1, x_2) to one party while sending (b, x_b) to the other. The seminal work of Ferret [59] provides an efficient implementation of ROT. Using ROT, P_1 and P_2 can securely generate multiplicative triples as follows:

1. P_1 and P_2 invoke $ROT()$ twice (with flipped roles) so that after the first call P_1 obtains (x_0, x_1) and P_2 obtains (b, x_b), whereas after the second call P_1 obtains $(b', x'_{b'})$ and P_2 obtains (x'_0, x'_1).
2. Let H be a random oracle with 1-bit output. P_1 computes and outputs

$$a_1 \coloneqq H(x_0) \oplus H(x_1); \quad b_1 \coloneqq b'; \quad c_1 \coloneqq a_1 \cdot b_1 \oplus H(x_0) \oplus H(x'_{b'}).$$

P_2 computes and outputs

$$a_2 \coloneqq H(x'_0) \oplus H(x'_1); \quad b_2 \coloneqq b; \quad c_2 \coloneqq a_2 \cdot b_2 \oplus H(x'_0) \oplus H(x_b).$$

It is easy to check that for all $b \in \{0,1\}$, $a_1 \cdot b_2 = H(x_0) \oplus H(x_b)$, and that for all $b' \in \{0,1\}$, $a_2 \cdot b_1 = H(x'_0) \oplus H(x'_{b'})$. Therefore, we have $(a_1 \oplus a_2) \cdot (b_1 \oplus b_2) = (c_1 \oplus c_2)$. The unpredictability of H's output, together with the security of ROT guarantees that P_1 (resp. P_2) learns nothing about (a_2, b_2, c_2) (resp. (a_1, b_1, c_1)).

3 Proposed Protocols

In this section, we elaborate on new different KDPI schemes, MT-DPI, CE-DPI, and BH-DPI, which have different initialization protocols but share the same AES-based traffic inspection scheme.

3.1 Session Initialization in MT-DPI

Rule delivery. For each keyword w_i, RG randomly generates a 128-bit r_i, and computes $\text{cert}_i = \text{cert}(w_i, r_i) = H(w_i, r_i)$. where H is modeled as a random oracle, hence evaluating H leaks no information on how H behaves on other input values. We instantiate H based on AES using the Davies-Meyer construction [57]: $H(x) = \mathsf{AES}_{\hat{k}}(x) \oplus x$. where $\hat{k}$ is a key picked by RG and known to both S and

Public Input: $\{\text{cert}_i\}_{i\in[K]}$ where $\text{cert}_i = \text{cert}(w_i, r_i) = H(w_i, r_i)$.
MB's Input: The keywords $\{w_i\}_{i\in[K]}$, certificate randomness $\{r_i\}_{i\in[K]}$.
S's Input: Handle generation key k.
Output: MB obtains $\{\eta_i\}_{i\in[K]}$ where $\eta_i = \eta(w_i) = H(k \oplus w_i)$.
Protocol:

1. For every $i \in [K]$,
   (a) Repeat the following s times where s is a statistical security parameter. In the j-th repetition:
      i. MB picks 128-bit uniform string r'_i and securely compute with S the function f

$$f(\{w, r', r\}; \{k, b\}) = H(mux(b, k \oplus w, (w, r)) \oplus mux(b, r', 0)$$

      where MB's secret input is w_i, r'_i, r_i and S's secret inputs are k and b_j.
   (b) For every $b_{i,j} = 1$, S learns $H(w_i, r_i)$, which is compared against the cert_i signed by RG (if they are not equal, S aborts immediately, reporting that MB was cheating).
   For every $b_{i,j} = 0$, S learns a value $H(k \oplus w_i) \oplus r'_i$. S verifies that the result is identical across all iterations with $j \in \{j \mid b_{i,j} = 0\}$ (otherwise, S aborts, reporting that MB was cheating).
   (c) S chooses any j such that $b_{i,j} = 0$ and sends the corresponding output $H(k \oplus w_i) \oplus r'_i$ to MB. MB then recovers η_i by XOR-ing the received value with r'_i.

Fig. 2. MT-DPI: Secure generation of handles for certified keywords (with malicious MB and semi-honest S)

MB. When AES is modeled as an ideal block cipher, even if $\hat{k}$ is not secret, H is still *collision-resistant* thanks to the security of Davies-Meyer.

RG then sends a signed message $\{(w_i, r_i, \text{cert}_i)\}_{i\in[K]}$ to MB and a signed message $\{\text{cert}_i\}_{i\in[K]}$ to S, where messages are signed with a standard digital signature scheme. Because H is collision-resistant and S does not know r_i, cert_i does not leak anything about w. In addition, we note that a malicious MB cannot forge a certificate for a word of his/her choice because S knows $H(w, r)$ and H is collision-resistant (MB cannot find a $(w', r') \neq (w, r)$ such that $H(w', r') = H(w, r)$).

Handle generation. We define a handle of a keyword w as $\eta_w = H(k \oplus w)$ where H is the same random oracle as described above and the key k is S's secret and w is MB's secret. MB and S will launch a secure two-party computation protocol to allow MB to obliviously learn the handle. However, without additional treatment, a malicious MB can obtain handles for any word $\omega \neq w$ of his/her choice and monitor the occurrence of ω in S's traffic. Next, we show how we prevent this attack using a cut-and-choose mechanism. Our full protocol is shown in Figure 2.

To obtain the handle of a keyword w, MB holding w, r', r and S holding k, b would jointly compute:

$$f(\{w, r', r\}; \{k, b\}) = H(mux(b, k \oplus w, (w, r)) \oplus mux(b, r', 0)$$

Public Input: Commitment of each bit of the keyword $\{c_j\}_{j\in[\ell]}$ where $c_j = g^{b_j}h^{y_j}$. Group elements g, h for computing the commitments.
MB's Input: The bit of keywords $\{b_j\}_{j\in[\ell]}$, commitment randomness $\{y_j\}_{j\in[\ell]}$,
S's Input: GC input labels $\{(X_j^0, X_j^1)\}_{j\in\ell}$ each corresponding to an input wire in the circuit
Output: MB obtains X_j^b, which can be used to evaluate the circuit
Protocol:

For every $j \in [\ell]$,

1. S chooses a random value r_j, and computes $e_j \coloneqq h^{r_j}$, $\sigma_j^0 \coloneqq H(c_j^{r_j})$, and $\sigma_j^1 \coloneqq H(c_j^{r_j} \cdot g^{-r_j})$.
2. S computes two envelopes: $E_j^0 = AES_{\sigma_j^0}(X_j^0)$, and $E_j^1 = AES_{\sigma_j^1}(X_j^1)$.
3. S sends E_j^0, E_j^1, e_j to MB.
4. MB computes $\sigma_j^b = H(e_j^{y_j})$. MB can decrypt $X_j^b = AES_{\sigma_b}^{-1}(E_j^b)$.

Fig. 3. CE-DPI: Keyword verification methods for one keyword (with malicious MB and semi-honest S)

where $mux(b, x, y)$ denotes evaluating a multiplexer on an input bit b and two same-length inputs x, y, which returns x if $b = 0$ and y if $b = 1$. That is,

$$f(\{w, r', r\}; \{k, b\}) = \begin{cases} H(k \oplus w) \oplus r' & \text{if } b = 0. \\ H(w, r) & \text{if } b = 1. \end{cases}$$

Since b is S's input, S controls whether the result will be a (masked) handle (by setting $b = 0$) or the certificate (by setting $b = 1$). For each w, the circuit will be evaluated s times, each with a freshly sampled b. S can then verify that for all $b = 1$, the outputs are all equal to $cert(w, r)$; and for all $b = 0$, the results all equal to an identical value $H(k \oplus w) \oplus r'$ where r' is the mask picked by MB. (Without the mask r', S will learn $H(k \oplus w)$ thus can launch selective probes to learn w.) Note that MB does not know b. So if MB used some $\omega \neq w$ in any of the s iterations, it will either fail to produce the correct certificate or output some inconsistent masked handles.

The cut-and-choose procedure above will be repeated s times. This statistical security parameter s directly impacts both overhead and security. The protocol cost is linear in s, and a malicious MB who tries to learn the token for a word different from a certified keyword will be caught with probability $1 - 2^{-s}$. It is standard to set $s = 40$, but in some settings it could be justified to use a smaller s. When MB cheats without being detected, it only gains the ability to scan for the presence of one word not in the keyword list. On the other hand, when MB is detected, the sender S learns that MB is trying to cheat, and can take action accordingly, such as not cooperating in traffic inspection, informing other parties, etc.

Security. The handle generation protocol given in Figure 2 is essentially a secure two-party computation protocol. Its security can be established following the real-world/ideal-world paradigm. Consider a malicious MB and semi-

honest S that interact in an ideal-world execution where a trusted third-party $\mathcal{F}_{\text{handle-gen}}$ exists:

1. $\mathcal{F}_{\text{handle-gen}}$ accepts k from S, and accepts (w_i, r_i) for all $i \in [K]$ from MB.
2. $\mathcal{F}_{\text{handle-gen}}$, with cert_i from RG, checks if $H(w_i, r_i) = \text{cert}_i$ for all $i \in [K]$. If the check passes, $\mathcal{F}_{\text{handle-gen}}$ sends $H(k \oplus w_i)$ to MB, and sends $\perp$ to S; otherwise, $\mathcal{F}_{\text{handle-gen}}$ sends "MB Cheats" to both MB and S.
3. If MB is honest, it outputs what was received from $\mathcal{F}_{\text{handle-gen}}$; otherwise, it can output anything. S always outputs what was received from $\mathcal{F}_{\text{handle-gen}}$.

A real-world protocol Π, where MB and S are the two interacting parties, is defined to be secure if for all $\text{cert}_i, (w_i, r_i), k$, the joint distribution of the outputs of MB and S is indistinguishable from that of the ideal-world execution.

If MB is honest while the sender S is compromised (but still semi-honest), we note that S cannot gain more information than every single run of the s repetitions of step (1a) because all s must be over exactly the same inputs (since S cannot deviate from the protocol to use different inputs across different runs of the circuit randomization sub-protocol). In this case, the whole protocol of Figure 2 simply reduces to a single run of Beaver's circuit randomization protocol and the security of our protocol can be derived directly from Beaver's protocol. Hence we have:

Theorem 1. *The handle generation protocol described in Figure 2 is a two-party computation protocol secure against a semi-honest S.*

If S is honest while MB is fully compromised, it is easy to see that the protocol is secure if MB does not deviate from the protocol. In case MB deviates from our protocol, its cheating behavior will be caught by step (1b) except for 2^{-s} probability, thus S outputting MB Cheats like in the ideal world.

Theorem 2. *The handle generation protocol described in Figure 2 is a two-party computation protocol secure against a malicious MB.*

Proof. For a malicious MB and honest S, we can construct a simulator $\mathcal{S}$ that is connected to the ideal-world $\mathcal{F}_{\text{handle-gen}}$ and interacts with MB as follows:

1. $\mathcal{S}$ runs the Figure 2 protocol as S to interact with MB.
2. If $\mathcal{S}$ aborts in step (1b), it generates random values and sends them to $\mathcal{F}_{\text{handle-gen}}$ as (w_i, r_i) (so these values cannot pass $\mathcal{F}_{\text{handle-gen}}$'s check of $H(w, r_i) = \text{cert}_i$).
   If $\mathcal{S}$ does not abort in step (1b), it extracts MB's effective input (w_i, r_i) through Beaver's circuit randomization protocol (note that Beaver's protocol offers input ex-tractability from an adversary) and sends them to $\mathcal{F}_{\text{handle-gen}}$.
3. $\mathcal{S}$ outputs whatever MB outputs.

A real-world execution (consisting of MB and S) will produce an indistinguishable output distribution from that of an ideal-world execution (consisting of the simulator $\mathcal{S}$ defined above, the sender S, and $\mathcal{F}_{\text{handle-gen}}$), because a deviating MB only gets to flip the values of any wires (including the input wires) in the Beaver's protocol to compute the handle, which will only result in one of the following consequences:

1. The value of $f(\{w, r', r\}; \{k, b\})$ is unaffected. In this case, MB's cheating behavior has no effect on the output distribution.
2. The value of $f(\{w, r', r\}; \{k, b\})$ is changed. In this case, however, for MB to successfully evade S's checks in step (1b), the results must be consistently changed for all $\{b_{i,j}\}_{j \in [s]}$, which happens only if MB guessed all s bits of b_j correctly, that is, with probability 2^{-s}.

Overall, the difference between the two output distributions are 2^{-s} at most.

Security against malicious S. It is easy to make our protocol also secure against a malicious sender by simply adding the following (rather symmetric) check in step (1b):

> For every $b_j = 1$, S sends all of $H(k \oplus w_i) \oplus r'_i$ back to MB, so that MB can verify that all these values are identical (otherwise, MB aborts and reports that S was cheating) before recovering $H(k \oplus w_i)$.

The security proof is also based on cut-and-choose, similar to what has been shown earlier against malicious MB. We did not include this treatment in our main protocol because in the context of KDPI, even if the handle generation protocol can handle malicious S, because a deviating S can easily sabotage other parts of the DPI system (such as the token computation step during the traffic inspection phase).

3.2 CE-DPI

CE-DPI (Commitment Envelope-DPI) combines garbled circuit protocol with Oblivious Commitment-Based Envelope (OCBE) [35] to ensure keyword integrity. Recall that in the DPI initialization phase, S constructed the garbled circuit and sends to MB. In the circuit, each bit of a keyword corresponds to an input wire of the circuit and has two associated secret values (one for 0 and one for 1). The protocol enables MB to learn the wire value corresponding to correct keyword bit, without S learning anything about the keyword.

In CE-DPI, RG stores cryptographic commitments for each bit of a keyword in a digitally signed certificate. For each keyword bit, S constructs two envelopes (encrypted messages) so that one can be decrypted when the committed value is 0, and the other when the committed value is 1.

More specifically CE-DPI uses the Pedersen commitment scheme [49]. To commit to a secret bit b, the committer generates a random value r, and publishes a commitment $c_i = g^b h^r$, where g, h are two generators of a group in which

discrete logarithm is hard. The verifier receiving c_i cannot learn the secret b without knowing the random value r.

CE-DPI uses an OCBE where the opening condition is that the committed bit equals to a pre-determined bit value. See [35] on how such an OCBE protocol works.

In CE-DPI, for each keyword w, RG computes a commitment for every bit of w:

1. RG computes $\{c_j\} = \{g^{b_j} h^{y_j}\}$ for all $j \in [\ell]$, where b_j is the j-th bit of w, and y_j is a fresh random value.
2. RG reveals $\{y_j\}_{j\in[\ell]}$ to MB and published a digitally signed $\{c_j\}_{j\in[\ell]}$.

S will verify the signed $\{c_j\}_{j\in[\ell]}$, and then construct two envelopes to deliver wire labels representing those bits to MB (Figure 3).

Since S sees only the commitments of the keywords, and the Petersen commitment scheme used in CE-DPI is information-theoretically hiding, S learns nothing about the keywords. it has been proven in [35] that ability to open the envelope not corresponding to the committed bit value implies the ability to compute the discrete log of $\log_g h$. Thus MB can obtain only the wire values corresponding to the keywords. Finally, note that S receives nothing from MB, it is easy to prove CE-DPI is secure against malicious MB and semi-honest S in the OT-hybrid model.

3.3 BH-DPI

For keyword integrity, BlindBox initially suggested RG to digitally sign each keyword, then use garbled circuits (or other 2PC protocols) to verify the signature without revealing the keywords. However, for any public-key signature scheme, implementing its verification function in garbled circuits is challenging and very expensive. At the time of writing this paper, we are not aware of any open-source code for garbled circuits support public-key signature verification.

Alternatively, one can use a cryptographic Batched Hash BH-DPI: (1) RG samples a secret random nonce, computes $h = H(w||\mathsf{nonce})$, signs h using a signature scheme, and sends nonce, h and the signature to MB. (2) MB sends S the value h with RG's signature. Later, S can verify h is indeed signed by RG, and use a garbled circuit to ensure that $h = H(w_1||w_2||...||w_n||\mathsf{nonce})$ (where H is modeled a random oracle and MB provides w_i and nonce) while the same w_i's are used for handle generation.

3.4 Traffic Inspection

The traffic inspection protocol for both MT-DPI and CE-DPI is described in Figure 4, enhancing BlindBox's in several respects.

More effective use of AES-NI. In BlindBox, the tokens for ω are computed as $T(\omega) = \mathsf{AES}_{\eta(\omega)}(ctr_\omega)$. The security property this achieves is that any adversary who knows multiple tuples in the form $(\omega, ctr_\omega, T(\omega))$ but does not know

Public Parameter: sliding window size ℓ. H modeled as a random oracle.
MB's Input: The list of keywords $\{w_i\}_{i\in[K]}$ and their handles $\{\eta_i\}_{i\in[K]}$.
S's Input: Handle generation key k and a stream of packets.
Output: MB learns every occurrence of w_i in the packet stream for all $i \in [K]$.
Protocol:

1. Initialize Counters and MB's token set:
   - S initializes a map M_{S} from ℓ-byte words to their counters (initialized to 0).
   - MB initializes a map M_{MB} from his list of keywords to their counters (initialized to 0).
   - MB computes the token set $TS \coloneqq \{T_i\}_{i\in[K]}$ where $T_i = H(\eta_i \oplus M_{\mathrm{MB}}[w_i])$.
2. For $j \in [N - \ell + 1]$, S moves its fixed-size sliding window to start at the j-th byte. Then,
   (a) Compute Tokens:
      - S computes $\eta \coloneqq H(k \oplus \omega)$, $ctr_\omega \coloneqq M_{\mathrm{S}}[\omega]$, $T \coloneqq H(\eta \oplus ctr_\omega)$ where ω is a word in current sliding window.

   (b) Match Tokens:
      - S sends T to MB.
      - If $T \in TS$ with $T = T_{i^*}$, MB learns w_{i^*} occurred at the j^{th} byte in the packet stream; otherwise, MB learns no keywords appeared at this location of the packet stream.

   (c) Update Counters and MB's token set:
      - S sets $M_{\mathrm{S}}[\omega] \coloneqq M_{\mathrm{S}}[\omega] + 1$.
      - If it was a match of keyword w_{i^*}, MB sets $M_{\mathrm{MB}}[w_{i^*}] \coloneqq M_{\mathrm{MB}}[w_{i^*}] + 1$ and $T_{i^*} \coloneqq H(\eta_{i^*} \oplus M_{\mathrm{MB}}[w_{i^*}])$; otherwise, MB do nothing.

Fig. 4. The traffic inspection protocol for MT-DPI, CE-DPI and BH-DPI

η_ω cannot monitor ω in the traffic. However, Their design fails to fully take full advantage of AES-NI [27] capabilities, since the tokens for different words are computed using AES with different keys. This means a key needs to be freshly scheduled per token, to introduce data-dependent stalls in the pipeline of AES-NI instructions.

To avoid such data-dependent stalls when running AES-NI instructions, we observe that the same security property desired for tokens can also be achieved by defining

$$T(\omega) = H(\eta(\omega) \oplus ctr_\omega)$$

where H is also modeled as a random oracle, defined the same as that used in MT-DPI's session initialization (Section 3.1), except that a different public fixed-key would be used to instantiate this random oracle.

Refreshing the counters. S will have to maintain a table of words that it has encountered so far and associate a counter with each word. The size of S's table is linear in the number of unique length-ℓ words in the traffic. As our experiments in Section 4 show, this table grows quickly. For example, over a 700MB traffic stream, we observed 7 million unique ℓ-byte words. It would be infeasible for S to efficiently maintain this large table of counters. Hence, S will periodically reset its counter table after inspecting a fixed amount (e.g., 100KB∼1MB) of

Table 2. Initialization costs in various network settings.

	Time cost (s)											B/W (MB)
	50 Mbps	100 Mbps	200 Mbps	300 Mbps	400 Mbps	500 Mbps	600 Mbps	700 Mbps	800 Mbps	900 Mbps	1 Gbps	
MT-DPI	135.6	109.2	95.2	91.2	90.4	89.2	67.2	66	65.2	64.3	64.3	425
BH-DPI	210.7	174.4	132.4	110.0	94.7	92.5	89.4	85.1	82.5	81.9	81.6	769
CE-DPI	181.1	122.5	112.4	100.8	96.7	93.6	91.5	89	88.2	87.8	87.6	530
BlindBox	817.9	455.6	273.4	212.0	180.4	163.7	151.7	142.3	137.0	132.2	132	4300

RTT for all network settings: 20ms. Bandwidth unit is MB.

traffic. Note that MB also needs to maintain a list of counters, one for each keyword. Thus, when S resets its table of counters, MB will do the same to stay synchronized on the keyword handles.

Comparing tokens. Using 16-byte tokens can be a waste of bandwidth. Instead, S could just send the first t bytes of the tokens to MB for comparison. The benefit is substantial bandwidth savings, which can be more than 50% if $t \leq 8$. The downside is that it will require an extra communication round between S and MB to avoid matching packets that contain the substring but not the actual keyword. It may also come with a potential information leakage of the keyword that MB learns extra information in the form of each occurrence of portions of the keywords. It is a trade-off between performance and efficiency.

4 Evaluation

In this section, we present a comparative evaluation of BH-DPI, CE-DPI, MT-DPI with existing protocols. We released our source-code at `https://github.com/mt-dpi`. In the appendix A, we discuss the integration of TLS.

4.1 Experiment Setup

Testbed. Our testbed consists of two machines (each has 8-core i7-3770 and 16GB memory) connected by a router, which supports 1Gbps Ethernet. We run S and MB on the two machines. As MB usually processes multiple connections, we experiment on several bandwidth capacities, ranging from 50Mbps to 1Gbps. We use Throttle [26] to configure both bandwidth capacities and RTTs.

Parameters. We assume that all the words and keywords are represented by 8-byte sequences, and set the length of a truncated token to 5 bytes. In the traffic inspection phase, the counters (of tokens) are refreshed every 1M tokens. We instantiate AES with AES-128 and implement it with `OpenSSL-1.1.1l` that supports AES-NI for fast token computation. We set $s = 40$ for 40-bit statistical security to run experiments for MT-DPI.

4.2 Initialization Phase

The cost of MT-DPI is mainly due to multiplicative triple generation and circuit evaluation. Since there is no data dependency across different invocations of AES

circuits, the entire circuits of $K \cdot s$ AES instances can be finished in 40 rounds (where K is the total number of keywords).

Thanks to the efficiency of Ferret OT [59] (about 0.5 bits and 4μs per random OT), the majority of the triple generation cost is due to constructing multiplicative triples from random OTs. The time cost of triple generation does not vary much across different network settings. However, the cost of triple generation exhibits a linear growth in terms of the statistical security parameter s, which determines the number of AES circuits to be securely evaluated.

Compared with CE-DPI and BH-DPI (see Table 2), MT-DPI uses less bandwidth, and we believe it is more suitable for network environments where real-world middle-boxes are deployed (RTT$\geq$20ms). Only when the network condition is ideal (very high bandwidth with at the same time very low round-trip latency), CE-DPI and BH-DPI could have some performance advantages.

We measure the initialization cost of BH-DPI and CE-DPI. These include (1) secure handle generation for the keywords, and (2) verification of the keywords. Table 2 shows that CE-DPI costs about 31.1% less bandwidth comparing to BH-DPI. Because of SHA3-256, 38,400 garbled AND gates must be computed per 15 keywords (960 bits); hence, 81.9K bytes of traffic is required per keyword. On the other hand, in CE-DPI, each keyword requires computing 1 group exponentiation and 2 additions, and sending one group element and two 16-byte envelopes. Thus, roughly 4.2KB bandwidth is used.

We also note that CE-DPI spends much time computing the envelopes whereas BH-DPI requires much time to transmit the garbled circuit. We find that as the connection speed increases, BH-DPI outperforms CE-DPI for connections over 350Mbps.

Comparison with existing protocols Compared with BlindBox, we find that the three new protocols are always better. That is because (1) BlindBox requires each word to be verified by one hash, while the verification parts are optimized in the new protocols; (2) the new protocols allow better utilization of AES-NI instructions. Compared with other ECC-based protocols, AES-based protocols do not have much benefits. However, the heavy cost in traffic inspection makes ECC-based protocols much less attractive and almost impractical.

4.3 Traffic Inspection Phase

We experimentally studied the benefit of AES-NI (used by MT-DPI, BH-DPI, and CE-DPI) for traffic inspection. Since MT-DPI, BH-DPI, and CE-DPI run identical traffic inspection algorithm, we use MT-DPI as a representative. The per-token costs (averaged over 1M tokens) are 67.78 ns for MT-DPI and 241.04 ns for BlindBox. That is, MT-DPI's traffic inspection runs 3.6x faster than BlindBox. BlindBox does not fully benefit from AES-NI because it uses a handle as an AES key, which not only requires running the more expensive key schedule frequently but also introduces data dependency that prevents the AES-NI instructions from leveraging a fully pipelined execution.

We compare the AES-based protocols (MT-DPI, BlindBox) with ECC-based protocols (e.g., PrivDPI, P2DPI and PE-DPI). For each token, ECC-based pro-

tocols require S to compute at least one group multiplication, which is far more expensive than AES operation. The cost for ECC-based protocols is at least 100x more expensive for traffic inspection phase than AES-based ones, which makes the ECC-based protocols impractical even if they allow leaner initialization.

5 Related Work

Several ECC based approach, such as PrivDPI [46], Kim et al. [30] , Pine [45], SEPM [7], ZKMB [23] can have fast initialization phase compared with AES-based approaches. However, the inspection phase are their main drawbacks which make them almost impossible to launch.

Several TLS extensions, including mcTLS [43], maTLS [33], and others [36, 39, 47], have been proposed for enabling middleboxes to perform DPI within TLS sessions. With these protocols, endpoints can authenticate middleboxes and grant them permission to read (or write) all (or part) of the packets. Unlike these TLS extensions, our scheme does not allow middleboxes to learn the plaintext.

Systems such as mbTLS [42], SGXBox [25], and others [24, 50], relied on trusted hardware like Intel SGX [1]. The basic idea is to let either one of the endpoints perform remote attestation to verify the middleboxes' integrity and sends cryptographic keys to the enclave. In comparison, our scheme does not require special trusted hardware.

There are other related approaches, such as Yuan et.al [61], BlindIDS [9], EndBox [21], SEST [15], Lai et.al [31], EV-DPI [51], PE-DPI [34], Embark [32], SplitBox [4], SPABox [19] and some machine learning approaches [58, 2, 6, 3] that try to analyze the encrypted contents to find some features. However, they either have a weaker threat assumption, (e.g., semi-honest MB, trusted Gateway, isolated two MBs) or have efficiency shortcomings (e.g., using curves to generate tokens, using look-up methods or using a public key to verify handles).

6 Conclusion

We studied efficient privacy-preserving keyword deep packet inspection protocols for network middleboxes. Through a close-up analysis of BlindBox[53], we proposed three new protocols MT-DPI, CE-DPI and BH-DPI that reduce the overhead and address vulnerabilities of lacking keyword verification. We implement and evaluate these protocols and compare them with BlindBox. Our experiment results show that they can outperform all prior protocols.

Acknowledgement

This work was supported by the KENTECH Research Grant(KRG202200048A).

References

1. Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative technology for CPU based attestation and sealing. In: International workshop on hardware and architectural support for security and privacy (2013)
2. Anderson, B., McGrew, D.: Identifying encrypted malware traffic with contextual flow data. In: ACM workshop on artificial intelligence and security (2016)
3. Anderson, B., Paul, S., McGrew, D.: Deciphering malware's use of tls (without decryption). Journal of Computer Virology and Hacking Techniques (2018)
4. Asghar, H.J., Melis, L., Soldani, C., De Cristofaro, E., Kaafar, M.A., Mathy, L.: Splitbox: Toward efficient private network function virtualization. In: Workshop on Hot topics in Middleboxes and Network Function Virtualization (2016)
5. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference (1992)
6. Blake, A., David, M.: Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. In: ACM International Conference on knowledge discovery and data mining (2017)
7. Bouscatié, E., Castagnos, G., Sanders, O.: Public key encryption with flexible pattern matching. In: International Conference on the Theory and Application of Cryptology and Information Security (2021)
8. Bujlow, T., Carela-Español, V., Barlet-Ros, P.: Independent comparison of popular DPI tools for traffic classification. Computer Networks (2015)
9. Canard, S., Diop, A., Kheir, N., Paindavoine, M., Sabt, M.: Blindids: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic. In: AsiaCCS (2017)
10. de Carné de Carnavalet, X., Mannan, M.: Killed by proxy: Analyzing client-end TLS interception software. In: Network and Distributed System Security Symposium (2016)
11. de Carné de Carnavalet, X., van Oorschot, P.C.: A survey and analysis of TLS interception mechanisms and motivations. arXiv e-prints (2020)
12. curl: curl: command line tool and library for transferring data with urls. `https://curl.se/` (1998)
13. curl: Curl: command line tool and library for transferring data with urls (1998)
14. Deri, L., Martinelli, M., Bujlow, T., Cardigliano, A.: NDPI: Open-source high-speed deep packet inspection. In: International Wireless Communications and Mobile Computing Conference (2014)
15. Desmoulins, N., Fouque, P.A., Onete, C., Sanders, O.: Pattern matching on encrypted streams. In: International Conference on the Theory and Application of Cryptology and Information Security (2018)
16. Dierks, T.: The TLS protocol version 1.2 (2008)
17. Durumeric, Z., Ma, Z., Springall, D., Barnes, R., Sullivan, N., Bursztein, E., Bailey, M., Halderman, J.A., Paxson, V.: The security impact of https interception. In: Network and Distributed Systems Symposium (2017)
18. Evans, D., Kolesnikov, V., Rosulek, M., et al.: A pragmatic introduction to secure multi-party computation. Now Publishers Inc (2018)
19. Fan, J., Guan, C., Ren, K., Cui, Y., Qiao, C.: SPABox: Safeguarding privacy during deep packet inspection at a middlebox. IEEE/ACM Transactions on Networking (2017)
20. Felt, A., Barnes, R., King, A., Palmer, C., Bentzel, C., Tabriz, P.: Measuring HTTPS adoption on the web. In: USENIX Security (2017)

21. Goltzsche, D., Rüsch, S., Nieke, M., Vaucher, S., Weichbrodt, N., Schiavoni, V., Aublin, P.L., Cosa, P., Fetzer, C., Felber, P., et al.: Endbox: Scalable middlebox functions using client-side trusted execution. In: IEEE/IFIP International Conference on Dependable Systems and Networks (2018)
22. Google: HTTPS encryption on the web. `https://transparencyreport.google.com/https/overview`, accessed: 2021-06-27
23. Grubbs, P., Arun, A., Zhang, Y., Bonneau, J., Walfish, M.: Zero-Knowledge middleboxes. In: USENIX Security (2022)
24. Han, J., Kim, S., Cho, D., Choi, B., Ha, J., Han, D.: A secure middlebox framework for enabling visibility over multiple encryption protocols. IEEE/ACM Transactions on Networking (2020)
25. Han, J., Kim, S., Ha, J., Han, D.: SGX-Box: Enabling visibility on encrypted traffic using a secure middlebox module. In: Asia-Pacific Workshop on Networking (2017)
26. Hedenskog, P.: Simulate slow network connections on linux and mac os x. `https://github.com/sitespeedio/throttle` (2021)
27. Hofemeier, G., Chesebrough, R.: Introduction to intel aes-ni and intel secure key instructions. Intel, White Paper (2012)
28. Jarmoc, J., Unit, D.: SSL/TLS interception proxies and transitive trust. In: Black Hat Europe (2012)
29. Khalife, J., Hajjar, A., Díaz-Verdejo, J.: Performance of opendpi in identifying sampled network traffic. Journal of Networks (2013)
30. Kim, J., Camtepe, S., Baek, J., Susilo, W., Pieprzyk, J., Nepal, S.: P2DPI: Practical and privacy-preserving deep packet inspection. In: AsiaCCS (2021)
31. Lai, S., Yuan, X., Sun, S.F., Liu, J.K., Steinfeld, R., Sakzad, A., Liu, D.: Practical encrypted network traffic pattern matching for secure middleboxes. IEEE Transactions on Dependable and Secure Computing (2021)
32. Lan, C., Sherry, J., Popa, R.A., Ratnasamy, S., Liu, Z.: Embark: Securely outsourcing middleboxes to the cloud. In: NSDI (2016)
33. Lee, H., Smith, Z., Lim, J., Choi, G., Chun, S., Chung, T., Kwon, T.T.: maTLS: How to make TLS middlebox-aware? In: NDSS (2019)
34. Li, H., Ren, H., Liu, D., Shen, X.S.: Privacy-enhanced deep packet inspection at outsourced middlebox. In: International Conference on Wireless Communications and Signal Processing (2018)
35. Li, J., Li, N.: OACerts: Oblivious attribute certificates. In: the Conference on Applied Cryptography and Network Security (2005)
36. Li, J., Chen, R., Su, J., Huang, X., Wang, X.: ME-TLS: Middlebox-enhanced TLS for internet-of-things devices. IEEE Internet of Things Journal (2019)
37. Lindell, Y., Pinkas, B.: A proof of security of yao's protocol for two-party computation. Journal of cryptology (2009)
38. Marquis-Boire, M., Dalek, J., McKune, S., Carrieri, M., Crete-Nishihata, M., Deibert, R., Omar Khan, S., Scott-Railton, J., Wiseman, G.: Planet blue coat: Mapping global censorship and surveillance tools (2013)
39. McGrew, D., Wing, D., Nir, Y., Gladstone, P.: TLS proxy server extension. `https://tools.ietf.org/html/draft-mcgrew-tls-proxy-server-01`
40. Moriarty, K., Morton, A.: Effects of pervasive encryption on operators. Tech. rep., RFC (2018)
41. Naylor, D., Finamore, A., Leontiadis, I., Grunenberger, Y., Mellia, M., Munafò, M., Papagiannaki, K., Steenkiste, P.: The cost of the "s" in https. In: ACM International Conference on Emerging Networking Experiments and Technologies (2014)

42. Naylor, D., Li, R., Gkantsidis, C., Karagiannis, T., Steenkiste, P.: And then there were more: Secure communication for more than two parties. In: the International Conference on emerging Networking EXperiments and Technologies (2017)
43. Naylor, D., Schomp, K., Varvello, M., Leontiadis, I., Blackburn, J., López, D.R., Papagiannaki, K., Rodriguez, P.R., Steenkiste, P.: Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In: ACM SIGCOMM Computer Communication Review (2015)
44. Nginx: Nginx. `https://www.nginx.com/` (2022)
45. Ning, J., Huang, X., Poh, G.S., Xu, S., Loh, J.C., Weng, J., Deng, R.H.: Pine: Enabling privacy-preserving deep packet inspection on TLS with rule-hiding and fast connection establishment. In: European Symposium on Research in Computer Security (2020)
46. Ning, J., Poh, G., Loh, J.C., Chia, J., Chang, E.C.: PrivDPI: Privacy-preserving encrypted traffic inspection with reusable obfuscated rules. In: ACM Conference on Computer and Communications Security (2019)
47. Nir, Y.: A method for sharing record protocol keys with a middlebox in TLS. `https://tools.ietf.org/id/draft-nir-tls-keyshare-02.html` (2012)
48. O'Neill, M., Ruoti, S., Seamons, K., Zappala, D.: TLS proxies: Friend or foe? In: the Internet Measurement Conference (2016)
49. Pedersen, T.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual International Cryptology Conference (1991)
50. Poddar, R., Lan, C., Popa, R.A., Ratnasamy, S.: SafeBricks: Shielding network functions in the cloud. In: USENIX Security (2018)
51. Ren, H., Li, H., Liu, D., Xu, G., Cheng, N., Shen, X.S.: Privacy-preserving efficient verifiable deep packet inspection for cloud-assisted middlebox. IEEE Transactions on Cloud Computing (2020)
52. Reports, V.: Deep packet inspection market size to reach usd 16620 million by 2026 at a cagr of 25.0 percent valuates reports (2021), `https://tinyurl.com/438yktzs`
53. Sherry, J., Lan, C., Popa, R.A., Ratnasamy, S.: BlindBox: Deep packet inspection over encrypted traffic. In: the ACM Conference on Special Interest Group on Data Communication (2015)
54. Silowash, G.J., Lewellen, T., Costa, D.L., Lewellen, T.B.: Detecting and preventing data exfiltration through encrypted web sessions via traffic inspection (2013)
55. Soghoian, C., Stamm, S.: Certified lies: Detecting and defeating government interception attacks against SSL. In: ACM Symposium on Operating Systems Principles (2010)
56. Waked, L., Mannan, M., Youssef, A.: To intercept or not to intercept: Analyzing TLS interception in network appliances. In: AsiaCCS (2018)
57. Winternitz, R.: A secure one-way hash function built from des. In: IEEE Symposium on Security and Privacy (1984)
58. Yamada, A., Miyake, Y., Takemori, K., Studer, A., Perrig, A.: Intrusion detection for encrypted web accesses. In: International Conference on Advanced Information Networking and Applications Workshops (2007)
59. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated ot with small communication. In: the ACM Conference on Computer and Communications Security (2020)
60. Yao, A.C.C.: How to generate and exchange secrets. In: Annual Symposium on Foundations of Computer Science (1986)
61. Yuan, X., Wang, X., Lin, J., Wang, C.: Privacy-preserving deep packet inspection in outsourced middleboxes. In: IEEE INFOCOM (2016)

A Integration with TLS

Table 3. Performance Evaluation of MT-DPI Integrated with TLS.

	RTT (S– MB, <1ms)			RTT (S– MB, 20ms)			RTT (S– MB, 200ms)		
	SplitTLS	BlindBox	MT-DPI	SplitTLS	BlindBox	MT-DPI	SplitTLS	BlindBox	MT-DPI
Elapsed Time to Retrieve Data (in ms)									
One Packet (1K)	0.68	3.35	2.87	40.88	42.77	42.42	400.79	403.28	402.72
One TLS Record (16K)	0.78	12.64	4.24	41.18	51.53	44.02	401.07	412.37	404.11
CPU Processing Overhead of S (in ms)									
One Packet (1K)	0.02	1.16	0.51	0.10	1.46	0.84	0.08	1.60	0.87
One TLS Record (16K)	0.10	9.11	1.97	0.13	9.46	2.14	0.10	9.31	2.22

Since TLS is the most widely deployed end-to-end protocol, we integrate MT-DPI into the TLS protocol by leveraging the TLS extension mechanism [16]. Our main focus is to make MT-DPI run within the TLS protocol so that it can be *easily and immediately deployed* without changes to applications.

We extended the TLS handshake protocol so that for S and R: (1) each has a copy of the key used to generate tokens, and (2) each establishes a separate TLS session with MB after authenticating the agreed MB between S and R. In addition, we instrument the TLS record protocol with the MT-DPI traffic inspection phase. We have verified that no modifications are needed to the applications by running our TLS extension with curl [12], Nginx [44] and Apache [13]). We compare MT-DPI with SplitTLS [28] and BlindBox [53]. We choose SplitTLS in our comparison because it is widely deployed in practice. For the size of the data that S sends, we select 1.5K and 16K as the former is one packet size of the Internet and the latter is the maximum length of the TLS message called the TLS record. The experiment results are shown in Table 3.

Elapsed time to retrieve data. We measure the elapsed time from the time when R requests a content from S to the time when R finally receives the content after the inspection by MB. The experimental results show that MT-DPI is faster than BlindBox in all the scenarios. With MT-DPI, the best scenario is that R receives a data from S for one TLS record in the 20ms RTT case. MT-DPI is 14.57% faster than BlindBox.

Interestingly, MT-DPI does not inflate the elapsed time for data retrieval compared with SplitTLS. It incurs only 3.77% of overhead for one packet and 6.90% for one TLS record that R receives from S with 20ms RTT.

CPU processing overhead. Since MT-DPI requires S to compute lots of tokens, it may incur high CPU processing overhead. To quantify the overhead, we evaluate CPU processing time while S gets a request message and sends a response message with the corresponding tokens. Our numerical results show that though MT-DPI requires relatively high processing overhead compared with SplitTLS, it reduces the overhead of BlindBox. The main reason is that our approach fully benefits from AES-NI, which makes the AES operation instantly completed. Finally, the processing time decreased.